MULTI-PATHS QUEST GENERATION FOR STRUCTURAL ANALYSIS

Mr. Thongtham Chongmesuk

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2018
Copyright of Chulalongkorn University

การสร้างเควสหลายทางเลือกสำหรับสตรัคเชอรัลอนาไลซิส

นายธงธรรม จงมีสุข

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

Thesis Title	MULTI-PATHS QUEST GENERATION FOR STRUCTURAL ANALYSIS
By	Mr. Thongtham Chongmesuk
Field of Study	Computer Engineering
Thesis Advisor	Assistant Professor Vishnu Kotrajaras

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirement for the Master of Engineering

-- Dean of the Faculty of Engineering

(Professor SUPOT TEACHAVORASINSKUN)

THESIS COMMITTEE

-- Chairman

(Associate Professor Setha Pan-ngum)

-- Thesis Advisor

(Assistant Professor Vishnu Kotrajaras)

-- Examiner

(Assistant Professor Sukree Sinthupinyo)

-- External Examiner

(Assistant Professor Jaratsri Rungrattanaubol)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

ธงธรรม จงมีสุข : การสร้างเควสหลายทางเลือกสำหรับสตรัคเชอรัลอนาไลซิส. (MULTI-PATHS QUEST GENERATION FOR STRUCTURAL ANALYSIS) อ.ที่ปรึกษาหลัก : ผศ. ดร.วิษณุ โคตรจรัส

ระบบสร้างเควสอัตโนมัติที่ใช้โครงสร้างเควสในปัจจุบันนั้นมีข้อจำกัด คือ ระบบไม่สามารถทำงานกับเกมที่มีสภาวะแวดล้อมแบบไดนามิค ซึ่งเป็นเกมที่การกระทำของผู้เล่นอาจส่งผลกระทบที่คาดไม่ถึงจากการตอบสนองอย่างต่อเนื่องของเกม เช่น เควสถูกล็อกอยู่ในสถานะที่ไม่สามารถจบได้ นอกจากนี้ ระบบยังไม่สามารถประกันได้ว่าเควสที่สร้างจะมีวิธีจบได้หลายวิธี

ในงานวิทยานิพนธ์นี้ โครงสร้างเควสที่ใช้การกระทำเป็นเป้าหมายถูกเปลี่ยนให้เป็นโครงสร้างเควสที่ใช้สถานะของเกมเป็นเป้าหมายโดยยังคงใช้กฎแกรมม่า เนื้อหาของเควสถูกสร้างด้วยการนำโครงร่างเควสมาใส่เนื้อหาด้วยระบบโทเคน ระบบโทเคนจะทำการคงความต่อเนื่องกันของเนื้อหาแต่ละชิ้นภายในเควส จากนั้นระบบสถานะเงื่อนไขสมบูรณ์จะสร้างชุดเป้าหมายเพิ่มเติมเพื่อช่วยป้องกันไม่ให้เควสถูกล็อก ระบบซักถามเควสทำการจำลองการเล่นในเกมที่มีสภาวะแวดล้อมแบบไดนามิค ระบบจะวิเคราะห์ทุกรูปแบบผสมของการกระทำที่เป็นไปได้เพื่อหาว่ามีกี่เส้นทางที่ผู้เล่นสามารถใช้จบเควสได้

เควสที่ใช้สถานะของเกมเป็นเป้าหมายนั้นมีความเข้ากันกับระบบจำลองเกมที่มีสภาวะแวดล้อมแบบไดนามิค ระบบซักถามเควสค้นพบเส้นทางที่ต้องใช้การวางแผนล่วงหน้าหลายขั้นพร้อมทั้งใช้ประโยชน์จากสภาวะแวดล้อมแบบไดนามิคเพื่อให้ทำเควสสำเร็จได้อีกด้วย ระบบที่ได้นำเสนอสามารถบรรลุวัตถุประสงค์และสามารถนำไปใช้เป็นโครงให้กับงานวิจัยต่อ ๆ ไปได้ อย่างไรก็ตาม ระบบจำลองเกมมีวัตถุในเกมจำนวนเพียงเล็กน้อยเนื่องจากระบบมีปัญหาด้านสมรรถนะ การศึกษางานนี้ในขั้นต่อไปควรเน้นไปที่การพัฒนาสมรรถนะของระบบ และปรับตัวระบบและระบบจำลองเกมให้สามารถใช้กับเกมเชิงพาณิชย์ได้

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

สาขาวิชา	วิศวกรรมคอมพิวเตอร์	ลายมือชื่อนิสิต ....
ปีการศึกษา	2561	ลายมือชื่อ อ.ที่ปรึกษาหลัก ....

5970176521 : MAJOR COMPUTER ENGINEERING
KEYWORD: Procedural content generation, quest generation, game state, commercial game, dynamic game world

Thongtham Chongmesuk : MULTI-PATHS QUEST GENERATION FOR STRUCTURAL ANALYSIS. Advisor: Asst. Prof. Vishnu Kotrajaras

Existing quest generation systems that used quest structures had an important limitation. They were not compatible with dynamic environment games, where player actions may have unforeseen repercussion from chain-reaction, such as locking the quest or preventing it from being completed. In addition, quests generated by such systems were not guaranteed to have multiple completion paths.

In this thesis, action-based-objective quests were replaced with game state-based-objective quests, while retaining the previous grammar rule. Each quest's contents and objectives were filled using Token system and Full Condition State system to ensure the connectivity within the quest's contents and to prevent players from locking the quest. The Quest Query system simulated a dynamic environment game. It analysed every combination of player's possible action to discover how many paths the player could complete the quest.

The state-based objectives were compatible with dynamic environment simulation. The query system discovered paths that used multiple steps of planning, including using the dynamic environment to complete the quest. The proposed systems functioned as intended and could be used as a framework for further studies. However, the simulation only used a small number of objects because of performance issue. Further study is advised to focus on improving the performance of the systems and applying the systems and its simulator to commercial products.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Field of Study:	Computer Engineering	Student's Signature ....
Academic Year:	2018	Advisor's Signature ....

ACKNOWLEDGEMENTS

ข้าพเจ้า ขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร. วิษณุ โคตรจรัส ซึ่งเป็นอาจารย์ที่ปรึกษาวิทยานิพนธ์ชิ้นนี้ ผู้ได้ให้คำแนะนำและชี้แนะแนวทางการวิจัย รวมทั้งคอยช่วยตรวจสอบจุดบกพร่องต่างๆจนกระทั่งงานวิจัยชิ้นนี้สำเร็จได้ด้วยดี งานวิจัยชิ้นนี้จะไม่สามารถสำเร็จได้เลยหากปราศจากความช่วยเหลือของอาจารย์

นอกเหนือจากอาจารย์วิษณุแล้ว ข้าพเจ้าขอขอบคุณคณะกรรมการ รศ.ดร.เศรษฐา ปานงาม ผศ.ดร.จรัสศรี รุ่งรัตนาอุบล และ ผศ.ดร.สุกรี สินธุภิญโญ ผู้ได้ชี้แนวทางในการเขียนงานวิจัยนี้ให้สมบูรณ์ยิ่งขึ้น

ผู้เขียนขอกราบขอบพระคุณ บิดา และ มารดา ของข้าพเจ้า ผู้ได้เลี้ยงดูและส่งเสริมสนับสนุนการศึกษาผู้เขียน ผู้เข้าใจและคอยให้กำลังใจผู้เขียนโดยตลอดมา

ขอขอบพระคุณคณะอาจารย์จุฬาลงกรณ์มหาวิทยาลัย ผู้ได้ประสิทธิประสาทวิชาแก่ผู้เขียนจนมีความรู้ความสามารถพอที่จะทำงานวิจัยนี้จนสำเร็จลุล่วงได้

ขอขอบคุณเพื่อนและพี่น้องร่วมห้องแลปแห่งตึกสี่ชั้นสิบแปด ณ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ผู้ให้คำแนะนำและช่วยเหลือทั้งการแก้ไขปัญหาทั้งทางเชิงเทคนิค และช่วยเหลือในการดำเนินเอกสารต่างๆของมหาลัย จนกระทั่งงานวิจัยนี้สำเร็จได้ด้วยดี

สุดท้ายนี้ผู้เขียนขอขอขอบคุณ คุณโยโกะ ทาโร่ (横尾 太郎) งานของคุณคือแรงบันดาลใจให้ข้าพเจ้าเริ่มศึกษาความเป็นไปได้ของเกมในฐานะของสื่อที่สามารถเล่าเรื่องที่ไม่มีสื่ออื่นสามารถเล่าได้ จนข้าพเจ้ามาถึงจุดๆนี้

Thongtham Chongmesuk

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

TABLE OF CONTENTS

Page

....iii
ABSTRACT (THAI)...iii
.... iv
ABSTRACT (ENGLISH) iv
ACKNOWLEDGEMENTS.. v
TABLE OF CONTENTS.. vi
1. Introduction...2
1.1. Objective 3
1.2 Scope of Work...3
2. Related Theories 5
2.1 Computer Role-playing Games and Quest 5
2.2 Quest..5
2.3 Procedural Content Generation (PCG) 6
2.4 Structural analysis of quest...16
2.5 Dynamic Game Environment 17
3. Related Works 20
3.1 Structural Analysis / Grammar Approach 20
3.2 Static quest in Generated game world 23
3.3 Dynamic and Nondeterministic Quests 24
3.4 Quest Monitoring and Adaptation...28
3.5 Quest Management and Evaluation 30

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

3.6 GameState Quest 34

4. Methodology 36

4.1 Overview 36

4.1.1 Quest Structure 36

4.1.2 Proposed Method Overview: 40

4.2 Action Rule table Modification 41

4.2.1 New Action Rule table = Component Table 44

4.3 Methodology 48

4.3.1 Creating blueprint for different type of quest. 48

4.3.2 Quest Generation 48

4.3.3 Token 51

4.3.4 Obtaining Restriction State 55

4.3.5 Obtaining Full Condition State 60

4.3.6 Checking for impossible quest 61

4.3.7 Path Finder / Quest Analysing 62

4.4.8 Avoiding infinite loop 67

5. Result and Analysis 71

5.1 Result 71

5.2 Comparison 74

5.2.1 Measurement and Evaluation Calculation 77

5.3 Path analysis 79

6. Summary 101

7. Future Work 104

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

REFERENCES 106

VITA 112

วิทยานิพนธ์

(THESIS)

ชื่อเรื่อง (ภาษาไทย) การสร้างเควสหลายทางเลือกสำหรับสตรัคเชอรัลอนาไลซิส

ชื่อเรื่อง (ภาษาอังกฤษ) Multi-Paths Quest Generation for Structural Analysis

เสนอโดย นายธงธรรม จงมีสุข

เลขประจำตัว 5970176521

หลักสูตร วิศวกรรมศาสตร์มหาบัณฑิต (ก2)

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

สถานที่ติดต่อ 45 ซอยเพชรเกษม 50 ถนนเพชรเกษม แขวงบางหว้า เขตภาษีเจริญ กทม. 10160

โทรศัพท์ 097 995 0515

อีเมล์ Thongtham.th.Chongmesuk@gmail.com
thongtham.c@student.chula.ac.th

อาจารย์ที่ปรึกษา ผศ.ดร.วิษณุ โคตรจรัส

คำสำคัญ (ภาษาไทย) การสร้างเนื้อหาอย่างอัตโนมัติด้วยวิธีการเป็นขั้นตอน การสร้างการผจญภัยอัตโนมัติ สถานะเกม เกมที่วางจำหน่ายในเชิงพาณิชย์ โลกของเกมที่เป็นพลวัต

คำสำคัญ (ภาษาอังกฤษ) Procedural Content Generation, Quest Generation, Game State, Commercial Game, Dynamic Game World

1. Introduction

Quest is one of the essential parts of Role-playing Games (RPG) genre. Quest informs players about what has to be done for the quest/story to go forward. Quest also records its state and recognizes when players had performed the required tasks.

Many computer RPG games' quests are too restrictive on what players can do to move the quest/story forward. They have very limited choices/paths to choose from. Still, building quests with multiple paths take a lot of design and development time. Thus, developers resort to do many single-path quests instead.

The lack of multi-path quests can affect replayability of games. Players usually get bored when playing any same type of quest for the second time since every action that must be done remains the same.

Developers also try giving only a goal state for a quest to allow players to discover the way to achieve the goal by themselves. However, in many cases there are usually only 1 specific action or a set of actions designed by developers that is guaranteed to lead player to that specific goal state. Other sets of actions that can finish the quest may or may not exist at all.

In order to deliver varied experience for each individual player, procedural quest generation (PQG) was proposed. PQG was a subset of procedural content generation (PCG), a system which created game contents automatically and non-deterministically. The main purpose of PCG is to present players with contents that change every time a game is played.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Still, current commercial PQG (in game such as Elder Scroll Skyrim [1], Fallout 4 [2]) generates quest by combining multiple pre-scripted single-path parts together to create a new linear quest. This guarantees that the generated quests are always consistent. But it limits the flexibility of player actions and quests content. The system is able to generate unique quests, but it cannot guarantee freedom of action of players on how to complete the quests.

This thesis introduces game state (or GameState) checking and action resolver into the quest generation which uses structural analysis. This new system replaces a part of quest generation that is responsible for generating a list of tasks/actions

(paths) a player character has to perform in order to reach a quest complete condition.

The new system can determine exactly how many paths a player can actually take to reach the same quest complete condition. The ability to measure player freedom of action should allow the quest generation system to generate quests with higher flexibility without compromising the integrity of the generated quests. The whole system will be called as Multi-Path Quest Generator, or MPQ-Generator.

Different players have different preferences and take different levels of enjoyment from a certain type of action. With increase in freedom of action on how to complete a quest, the quest has higher potential to entertain players due to higher possibility that players can find actions that give them maximum enjoyment. The new algorithm developed in this proposal aims to introduce a new tool and knowledge which can be implemented into commercial games.

1.1. Objective

- This thesis aims to develop a quest generation system that considers dynamic aspect of the game and generate quests around it.
- The quest generation system will use new quest structure that can work with dynamic environment system while retaining the advantage of Grammar Approach.
- The quest generation system can generate quests with specific number of completion paths according to user configuration.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

1.2 Scope of Work

- The MPQ-Generator will only focus on Non-Player-Character (NPC) to Non-Player-Character interaction. not NPC-to-Environment interaction. An example of NPC-to-Environment interaction is in stealth game where NPC will know that a player has murdered a character when the body (which is now an environment object) is found by it or another NPC.

- MPQ-Generator does not consider current active quest during quest path finding. This means that information such as “by doing Path-A to complete quest X, quest Y will also be completed alongside it” is not available in the path’s information.
- The generated quests are tested in a simulator, not in actual commercial games.
- Quality improvement will not be applied to the generated quests.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

2. Related Theories

2.1 Computer Role-playing Games and Quest

Role-playing Games (RPGs) is a game genre that originated from pen and paper fantasy wargame. The term ‘role-play’ can be tracked back to historical re-enactment or ‘Theatre Game’ where each actor had to act and improvise based on the character the actor was ‘role-playing’. In RPGs, players control character (one or multiple) to explore the game world and interact with it. One of the main goals of RPGs is to ‘interact’ with the game world and observe how the game world reacts to players’ actions.

The term RPGs can also be used to a describe a game which has a progressive development mechanic in player’s character ability and equipment; such as unlocking an ability to fly or upgrading weapon to deal higher damage.

RPGs’ main element is narrative, exploration, strategic planning, and deep character interaction, rather than combat or precision timing. However, other genres may implement RPGs element to create sub-genre. For example, Action-RPGs such as Dark Souls series emphasize real-time combat, and strategy-RPGs such as Final Fantasy Tactics series and Crusader King series emphasize planning and actually play closer to chess.

The narrative of RPGs’ story can be dynamic or static based on game story design. However, most RPGs’ stories have a main storyline which the whole narrative revolves around. This could be ‘saving your kingdom from Alien invasion’, ‘seeking your missing parent’, or ‘revealing the mystery of a certain anomaly’.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

2.2 Quest

Quests in computer games comes in many definitions depending on context, focus, and type of game. Quests in RPGs and First-Person-Shooters are different. Similarly, quests in Adventure games and Platforming games are different. In this thesis, a quest is defined as “the array of soft rules that describes what the player has to do in a particular storytelling situation” [3]. A quest has the following properties.

- **Storytelling:** A quest must be based on motivation or narrative background within the game world.
- **Obstacle:** Task(s) must be completed to finish a quest.
- **Reward:** Finishing a quest will give reward to the quest-receiver, be it item or story advancement.

Story-wise, quests can be categorised into 2 groups, main-quest and side-quest. A main-Quest revolves around the 'main' storyline of the game, while a side-quest usually revolves around sub-plot or something completely non-related to the story. Gameplay-wise, quests can be categorised into 9 groups [4]. Kill Quest, Loot Quest, Gathering Quest, Boss Kill Quest, Escort Quest, Interaction Quest, Exploration Quest, Delivery Quest, and Defend Quest.

2.3 Procedural Content Generation (PCG)

Procedural Content Generation (PCG) is a system which creates contents for player consumption automatically as players play the game. The content can be generated by selecting a static pre-determined object, or combining multiple objects together to create a more unique content. The content can range from new items, new enemies, new non-playable characters (NPC), new abilities, to new areas. An example of PCG is map randomizer in Spelunky [5]. Shown in Figure. 1 are multiple maps of Spelunky. A map will be generated when players finish a level and enter the next level. Each player will encounter different maps even when he/she enters the same level (Jungle in Figure.1).

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

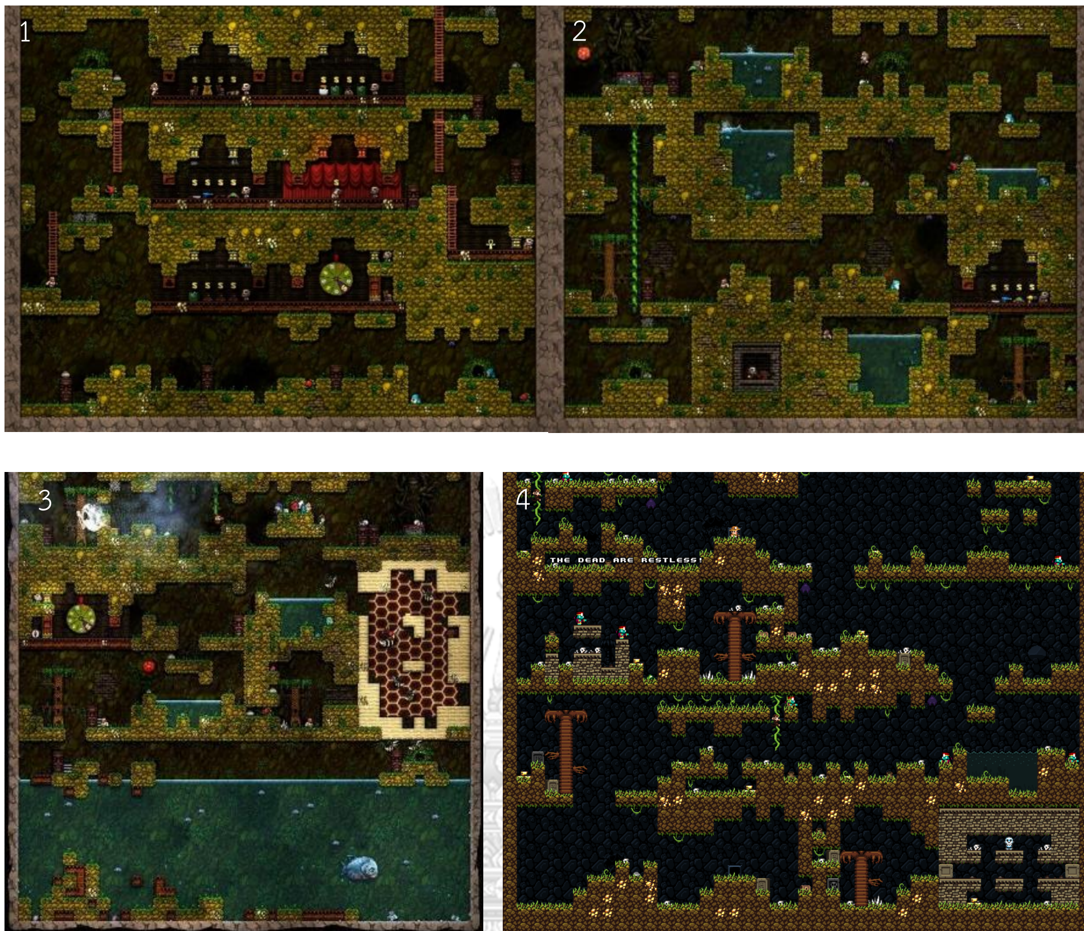


Figure. 1: A collection of maps generated for Spelunky's jungle area. Bottom right map (4) is taken from the classic graphic version. Images taken from

(1: http://spelunky.wikia.com/wiki/The_Jungle/HD [6])

(2: http://spelunky.wikiam.co/wiki/Black_Market [7])

(3:https://www.reddit.com/r/spelunky/comments/
521rql/looking_for_highres_images_of_full_levels/ [8])

(4: http://spelunky.wikia.com/wiki/Restless_Dead_level [9]).

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

PCG allows games to create a 'unique' experience for each individual player using the random nature of PCG, thus it increases the replayability and values of those games. PCG also reduces the burden of game developers by lowering the amount of manual labour the developers have to do.

Procedural Quest Generation (PQG) is a subset of PCG and can be generalised into 2 categories, space and plot automation. Figure. 2 shows 4 different

styles of generation from these categories, taken from B.Kybartas and R. Bidarra's "A survey on story generation techniques for authoring computational narratives" [10].

Space includes game world objects and environments, such as items, NPCs, geometry of the game world, weather, and such. Low space automation may result in only variety in enemy placement or randomness in reward from treasure chests. High space automation may generate a whole city with random NPCs for players to explore and perform tasks to complete a quest.

Plot is the non-tangible part of content which dictates how the player and the tangible part interact with each other. Using an assassination event in a storyline as an example, low plot automation may randomize only the place the murder happens. Medium automation may vary the way the assassination is committed, but the assassination happens nonetheless. High automation may result in an event that the victim actually survives and changes the story that happens afterwards.

The level of each type of automation, space and plot, can be used to identify the type of content that could be generated from the PQG system. PQG can be used in a wide range of genres and narrative styles. Table. 1 and Figure. 3 to Figure 15 show a few examples of games which implement different levels of automation, together with their screenshots.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

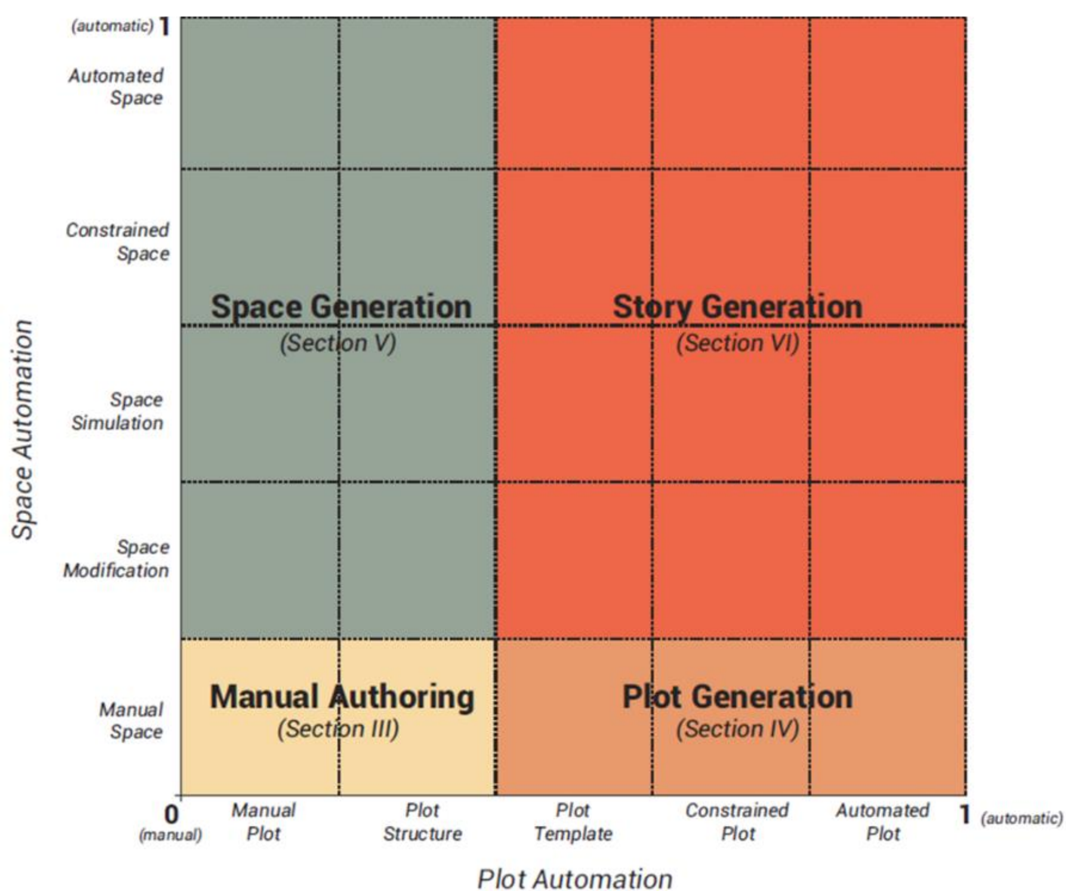


Figure. 2: Graph taken from B.Kybartas and R. Bidarra, "A survey on story generation techniques for authoring computational narratives" [10].

CHULALONGKORN UNIVERSITY

Table. 1: List of Games with different level of automation according to Figure. 2's graph.

Space Generation	Story Generation
-Spelunky [5] -Neo Scavenger [11] -Renowned Explorers: International Society [12]	-Dwarf Fortress [13] -Rimworld [14]
Manual Authoring	**Plot Generation**
-The Witcher 3: Wild Hunt [15] -Nier: Automata [16] -World Of Warcraft [17]	-Mount&Blade Warband [18] -Middle Earth: Shadow of Mordor [19] -The Elder Scroll:Skyrim [1] -Fallout 4 [2]

<u>Screenshot of games in Space Generation Category</u>

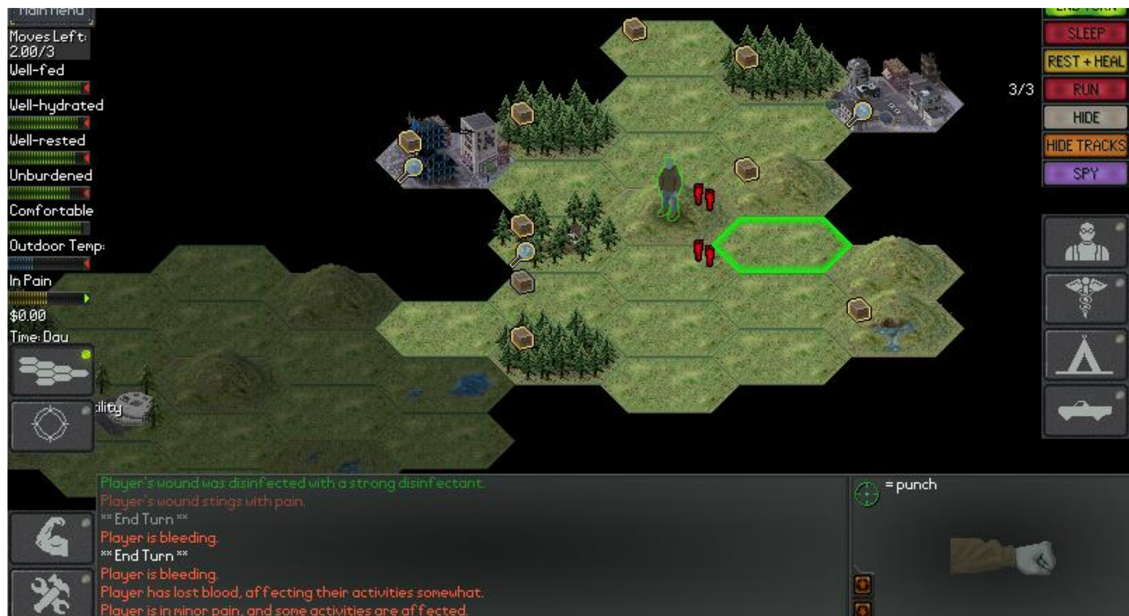


Figure. 3: A screenshot of Neo Scavenger, a turn-base simulation role-playing game. Image. downloaded from http://bluebottlegames.com/games/neo-scavenger [20].

Figure. 4: A screenshot of Spelunky, a real-time action-platforming game. Image downloaded from http://www.spelunkyworld.com/images/spelunky-ss01.jpg [21].

Figure. 5: A screenshot of Renowned Explorers: International Society, a turn-base tactical role-playing game. Image downloaded from http://renownedexplorers.com/#screenshot-ec-2 [22].

Screenshot of games in Manual Authoring Category

Figure. 6: A screenshot of The Witcher 3: Wild Hunt, an open world role-playing game. Image downloaded from https://gamesdb.launchbox-app.com/games/images/15977 [23].

Figure. 7: A screenshot of Nier: Automata, an action-RPG. Image downloaded from https://www.niergame.com/gb/ [24].

Figure. 8: A screenshot of World of Warcraft, a massively multiplayer online RPG (MMORPG) by Activition-Blizzard.

Image downloaded from http://www.pcgamer.com/what-we-want-from-world-of-warcraft-in-2017/ [25].

Screenshot of games in Story Generation Category

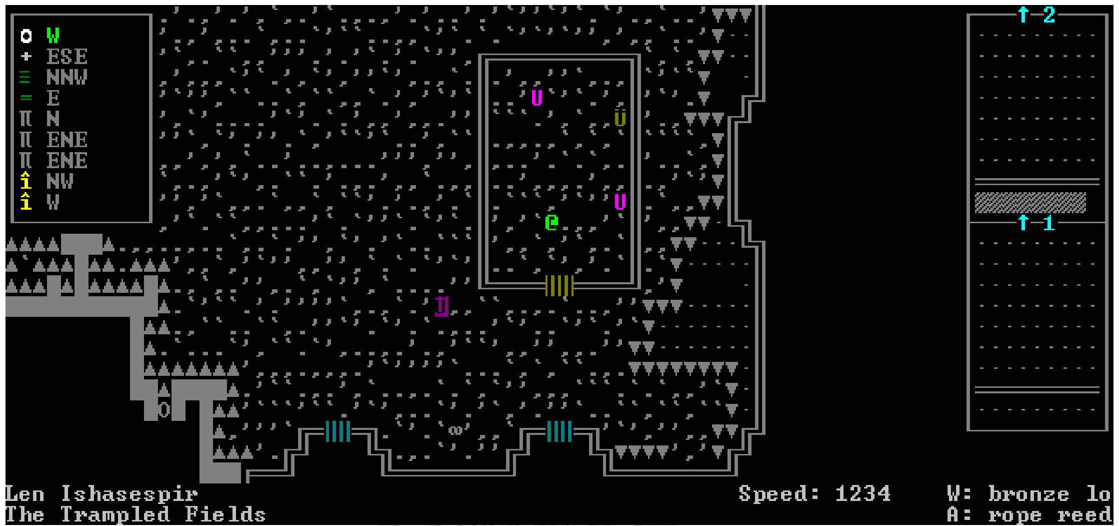


Figure. 9: A screenshot of Dwarf Fortress's text-based graphic, a semi-real-time strategy simulation game. Image taken from http://www.bay12games.com/dwarves/screens/adv1.html [26].

Figure. 10: A screenshot of a user modification of Dwarf Fortress's graphic engine to render an isometric 2.5D graphic. Image taken from https://www.polygon.com2014775877073dwarf-fortress-3d-mod [27].

Figure. 11: A screenshot of Rimworld, a semi-real-time strategy simulation game. Image downloaded from https://rimworldgame.com/images/screens/megacolony.jpg [28].

Screenshot of games in Plot Generation Category

Figure. 12: A screenshot of Mount&Blade Warband, a medieval simulation role-playing game. Image downloaded from https://www.heypoorplayer.com/2016/10/01/mount-blade-warband-gets-feature-video/ [29].

Figure. 13: A screenshot of Middle Earth: Shadow of Mordor, an action-RPG. Image downloaded from https://segmentnext.com/2014/09/30/understanding-nemesis-system-in-middle-earth-shadow-of-mordor/ [30].

Screenshot of games in Plot Generation Category

Figure. 14: A screenshot of The Elder Scroll: Skyrim, a role-playing game. Image downloaded from https://www.digitaltrends.com/game-reviews/the-elder-scrolls-v-skyrim-review/ [31].

Figure. 15: A screenshot of Fallout 4, a shooting role-playing game. Image downloaded from https://www.gamecrate.com/fallout-4-radpacks-horrors-commonwealth/13791f [32].

From Table. 1, Notable examples of PQG are “The Elder Scroll: Skyrim” and “Fallout series”, which use ‘Radiant AI’ from Bethesda Softworks. Both The Elder Scroll and Fallout have static game worlds where most objects are hand crafted and placed. The Radiant AI system is a PQG system which generates quests for players. A generated quest will have fixed task and narrative, but the object that the players must interact to complete the quest will be randomly chosen from the objects in the game world.

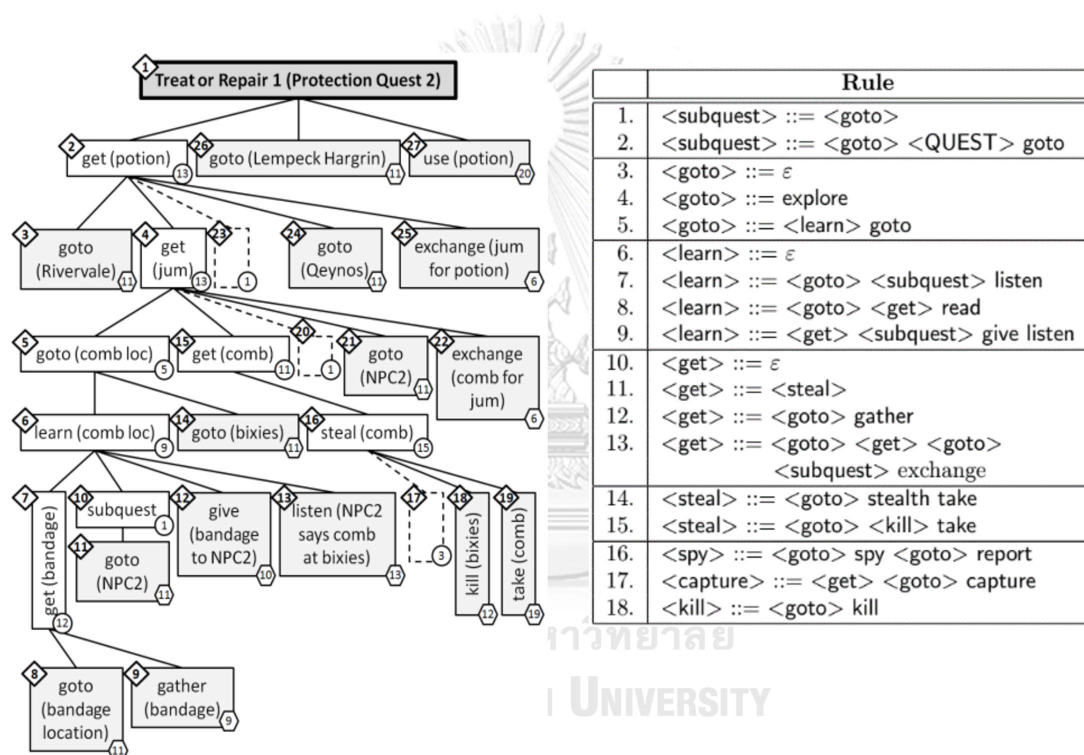
Further information of each game can be found on [1, 2, 5, 11-19].

2.4 Structural analysis of quest

Structural Analysis approach in quest generation is a way to construct quest in similar approach to constructing a sentence using ‘common’ grammar. The ‘grammar’ and ‘vocabulary’ rules of structural analysis were created by classification, analysis, and dissection of quests from multiple RPGs to get a common pattern which all quests shared. Quests were generalised into ‘motivations’ (distinct

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

underlying drives that compel each quest/narrative to happen). For each ‘motivation’, a quest could be categorised into different ‘strategies’, the outlines on how the quest (motivation) could be completed. Finally, each ‘strategy’ was linked to specific set of ‘Sequence of Actions’ which described general tasks (actions) the player or NPC needed to perform to complete the quest. The tasks (actions) were usually in the Action rule (<ACTION>) form, which could be broken down into specific Atomic Action (ACTION) depending on the Action Rule table. Figure. 16 shows an example of quest structure generated by structural analysis.



	Rule
1.	<subquest> ::= <goto>
2.	<subquest> ::= <goto> <QUEST> goto
3.	<goto> ::= ε
4.	<goto> ::= explore
5.	<goto> ::= <learn> goto
6.	<learn> ::= ε
7.	<learn> ::= <goto> <subquest> listen
8.	<learn> ::= <goto> <get> read
9.	<learn> ::= <get> <subquest> give listen
10.	<get> ::= ε
11.	<get> ::= <steal>
12.	<get> ::= <goto> gather
13.	<get> ::= <goto> <get> <goto> <subquest> exchange
14.	<steal> ::= <goto> stealth take
15.	<steal> ::= <goto> <kill> take
16.	<spy> ::= <goto> spy <goto> report
17.	<capture> ::= <get> <goto> capture
18.	<kill> ::= <goto> kill

Figure. 16: An example of quest (left) generated using Action Rule table (right), taken from [33].

2.5 Dynamic Game Environment

The changing of game environment and conditions are not limited to player actions. Some games allow interaction between objects, such as an NPC’s routine action or the growing of plants, which can happen automatically without the need to get the player’s involvement. For example, when an NPC health is low, he/she may seek doctor to recover his/her lifepoints.

One way to produce such dynamic world is to use quests. Quests are not exclusive to players. NPCs can also do their own quests. NPC's quests make each NPC behaviour more realistic. A game can create NPC-only quests such as 'bear hunting' for NPCs who perform hunter job in the game world. Instead of spawning bear meat directly into the NPCs inventory, the NPCs have to actually complete the bear hunting quest in order to obtain the meat. John Grey and Joanna Bryson's "Procedural quests: A focus for agent interaction in role-playing-games" [34] proposed this type of dynamic interaction.

A good example of players sharing a quest pool with NPCs can be found in the original build of S.T.A.L.K.E.R, which is now known as S.T.A.L.K.E.R.: Shadow of Chernobyl. In that original build, NPCs had the same set of available actions similar to those of players, along with AI system that allowed them to perform those actions. This system was sound when looking into the game story, the player was one of many S.T.A.L.K.E.R.s in the area, and anyone (player or NPCs) was able to be 'THE ONE' who solved the mystery littering around the game world. This system allowed players to encounter different 'story' depending on how each player and NPCs chose to do their quests. However, the GSC Game World (developer of S.T.A.L.K.E.R.) scrapped the system because it was not fun for players. Testers found that they were locked out of contents and quests because NPCs had already finished those quests [35]. Most importantly, NPCs were able to finish the main story quest before the players could and ended the game prematurely. This was one of the problems when dynamic Questing NPCs system was used in a game which had limited set of quests.

Dynamic world had become an essential selling point for many commercial games, notably Elder Scrolls: Skyrim, Fallout 4, Kingdom Come: Deliverance, and Mount&Blade series. In these games a player action could cause a chain reaction that allowed many new ways to achieve the same goal. For example, if a player wanted to put NPC A in jail as part of a quest, and the player knew that the police NPC would put any 'character' in jail if he/she committed crimes, then the player could give NPC A some illegal substance and alerted the police. Or the player could put a criminal jumpsuit inside the NPC's bag of clothes. NPCs must always wear clothes in

his possession, according to the rule. Thus, NPC A had to wear the criminal jumpsuit. When a police NPC saw NPC A in the criminal jumpsuit, the police NPC would arrest NPC A on sight according to the game rule. This was not realistic, but it was viable due to the way game mechanics and rules interact with each other. Good dynamic system would allow players to manipulate dynamic nature of the game world to achieve their goals without the need for developer to explicitly script such events.

3. Related Works

New ways to generate quests for computer games are being researched and developed. One such work is called "structural analysis" by Doran and Parberry [33]. Their quest analysis from 4 MMORPG games concluded a common structure of quests which could be used in quest generation. Other works in quest generation that used different approaches include Hierarchical Generation of Dynamic and Nondeterministic Quests in Games [36], and Dynamic Quest Plot Generation using Petri Net Planning [37]. In these works, dynamic quests and/or dynamic game environment were important aspects in quest generation.

3.1 Structural Analysis / Grammar Approach

Based on "A Prototype Quest Generator Based on a Structural Analysis of Quests from Four MMORPGs" [2011] by Doran and Parberry [33], Structural Analysis approach in quest generation was a technique to construct quest using 'common' grammar. The 'grammar' and 'vocabulary' rule of structural analysis was created by classification, analysing, and dissecting quests from multiple RPG games to get common patterns shared by quests. Quests were categorised according to 'motivation', the distinct underlying drive (narrative) that compelled the quest. Within each 'motivation', the quest could be categorised into different 'strategy', the outline on how the quest (motivation) can be completed (satisfied). The motivations were Knowledge, Comfort, Reputation, Serenity, Protection, Conquest, Wealth, Ability, and Equipment (see Table. 2).

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Table. 2: A Motivation table. Taken from [33], Page 4, Table 4

Motivation	Strategy	**Sequence of Actions**
Knowledge	**Deliver item for study** **Spy** **Interview NPC** **Use an item in the field**	<get> <goto> give <spy> <goto> listen <goto> report <get> <goto> use <goto> <give>
Comfort	**Obtain luxuries** **Kill pests**	<get> <goto> <give> <goto> damage <goto> report
Reputation	**Obtain rare items** **Kill enemies** **Visit a dangerous place**	<get> <goto> <give> <goto> <kill> <goto> report <goto> <goto> report
Serenity	**Revenge, Justice** **Capture Criminal(1)** **Capture Criminal(2)** **Check on NPC(1)** **Check on NPC(2)** **Recover lost/stolen item** **Rescue captured NPC**	<goto> damage <get> <goto> use <goto> <give> <get> <goto> use capture <goto> <give> <goto> listen <goto> report <goto> take <goto> give <get> <goto> <give> <goto> damage escort <goto> report
Protection	**Attack threatening entities** **Treat or repair (1)** **Treat or repair (2)** **Create Diversion** **Create Diversion** **Assemble fortification** **Guard Entity**	<goto> damage <goto> report <get> <goto> use <goto> repair <get> <goto> use <goto> damage <goto> repair <goto> defend
Conquest	**Attack enemy** **Steal stuff**	<goto> damage <goto> <steal> <goto> give
Wealth	**Gather raw materials** **Steal valuables for resale** **Make valuables for resale**	<goto> <get> <goto> <steal> repair
Ability	**Assemble tool for new skill** **Obtain training materials** **Use existing tools** **Practice combat** **Practice skill** **Research a skill(1)** **Research a skill(2)**	repair use <get> use use damage use <get> use <get> experiment
Equipment	**Assemble** **Deliver supplies** **Steal supplies** **Trade for supplies**	repair <get> <goto> <give> <steal> <goto> exchange

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Table 2 shows Strategies for each of the motivations and their respective Sequence of Actions. ‘Sequence of Actions’ described tasks (actions) the players or NPCs had to perform to complete the quest. The tasks (actions) were usually in the Action Rule form, which could be broken down into specific Atomic Action depending on the Action Rule table. (see Figure. 16 for Rule table).

Their work used these structures to create a prototype quest generation system. The prototype quest generation used Prolog language because of its ability to backtrack and try new solution (breaking down <Rule> and filling in quest details).

Doran and Parberry’s quest generation system did not address the coherence between multiple generated quests. Buss,D.B. and his peers [38] tried to address this by introducing the system they called Quality System, a progressive tier system. They used structural analysis from [33] as a baseline and implemented the Quality System to increase the cohesiveness and connectivity between generated quests. The progressive tier system introduced 2 new metrics; NPC tier and NPC profession. NPC profession determined the type of quests and contents (motivations, strategies, items, enemies, and such) available from each NPC. While NPC tier determined the level of quest that would be generated associated with each NPC. When the 2 metrics were combined, it allowed the system to determine the appropriate quests available from an NPC. For example, a farmer NPC who had ‘farmer profession’ and ‘minor NPC’ could not use the motivation ‘Serenity’ in Table 2 to generate a quest and could not use ‘dragon’ enemy in his quest. A king NPC who had ‘ruler profession’ and ‘epic NPC’ was able to use both motivation and enemy, but could not use strategy ‘check on NPC’ under ‘Serenity’ motivation, which was below its ‘epic’ tier.

This quest structure / ‘grammar’ was further expanded in Machado, Santos and Dias’ work [39]. In their work, quests from a single-player game (The Witcher 3) were analysed. They found differences in quests between The Witcher 3 and other MMORPGs. The quest structures were adjusted and modified, new Action Rules and Atomic Actions were introduced in order to cover a boarder range of quests. This allowed for more expressive quests and more variety in the quests.

MPQ-Generator, the quest generator proposed by this thesis, used the same quest structure in its quest generation. However, the Action Rule table and the

breakdown of Action Rule to Atomic Action were modified to function with dynamic environment aspect supported by MPQ-Generator.

3.2 Static quest in Generated game world

Procedural generated narrative is not achieved only by generating a quest suitable for a presented world, but also by generating a world to suit a presented quest. Calvin Ashmore and Michael Nitsche [40] presented a procedural game world with quest generator. In their work, a 'quest' was a set of 'locks' node and 'keys' node that must be accessed in certain order, similar to a puzzle maze with locked doors. For example, a lock 'river' required a key 'swimming skill' in order to unlock (pass through) and reach the next portion of the 'quest'. When a quest was generated with its associated game world, the game world was generated tile by tile. Locks could be placed at the edge of these tiles.

Ashmore and Nitsche implemented a graph representing the relationship between locks and keys to create a coherence game world and avoid unreachable pair of lock and key (placing a key behind its corresponding lock). A node of the graph represented game space. It informed the system how the game world was to be generated.

The other attempt in this type of PGC was by Valls-Vargas and peers [41]. Their work focused on generating a map that could support multiple quests/stories that unfolded non-linearly. The system used a collection of plot points (e.g. 'The player meets Henry') as input to generate a map for the given narrative.

This thesis differed greatly from both works because MPQ-Generator's quest generation did not generate a map to match generated quests.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

3.3 Dynamic and Nondeterministic Quests

Soares de Lima, Feijó, and Furtados' work [36] introduced new algorithm and model for constructing quest in real-time, along with dynamic narrative and multiple endings. A quest was decomposed into a 'primitive-quest' (main quest frame) with/without multiple 'sub-quests' inside. The dynamic nature of this algorithm came from the ability to switch 'sub-quest' in/out depending on the current game state. If a 'primitive-quest' was about delivering an antidote to a patient, a 'sub-quest' could be described as "goto hospital A> picking antidote > goto patient > apply antidote". However, if the antidote was destroyed during the "goto" point, a 'sub-quest' "goto hospital B > picking antidote" could be injected into that point in real-time, to prevent the quest from failing. Each 'sub-quest' had conditions that must be met before it could be selectable from the quest pool. For example, the 'picking antidote' sub-quest was only selectable if an antidote object existed in the game world and it was reachable by the player.

In addition, instead of generating the whole quest with all possible combinations of sub-quests, the algorithm generated the quest as the player carried out 'action' after the main quest was generated and monitored which 'sub-quest' was best injected into the main quest. Every time a player performed an action, the system analysed the current situation and conditions of the game and generated a new part of the quest, if needed. This process lowered the time required for quest generation significantly compared to generating all possible sub-quests at the start of the generation.

A similar approach to generate quest solutions was developed by Imran Khaliq and Zachary Watso [42] in "The Omni Framework". However, instead of continuously updating the quest's objectives and actions that must be taken, the system generated quests' objectives to match players' preferred playstyle and current game state. The framework oversaw the whole game world and kept records of everything that had happened and is happening within the game, including meta data such as completed quests or player's actions. The framework separated the recorded information into 3 categories which affected different parts of quest generation, Player System, Non Player Character System, and World State System.

There were also 2 additional systems that governed the quest generation directly, Basic Quest System and Destiny System.

Omni Framework was intended to be used from the start of the game to gather as much data about players as possible. During the initial state where the framework had little information regarding the players, the Basic Quest System would generate simplistic quests. When more information became available, more complex quests that matched specific playstyle became selectable during quest generation. The framework recognised 2 major playstyles, aggressive and non-aggressive. Quests would be generated differently for each playstyle. When a quest was generated, the framework consulted the Player System, NPC System, and World State System to determined which ‘solution templates’ were active and selectable. Then a random amount of the solution templates would be selected. The ‘solution template’ also contained both step-of-actions that must be taken and the quest rewards (effect). The possible solutions or quests objectives were updated according to player actions and the new game state.

While Basic Quest System generated isolated quests, the Destiny System generated chain of quests that were inter-connect with one another. When a new ‘Destiny Node’ (Destiny quest) was generated, it recognised objects involved with previous Destiny quests and tried to make the generating quest use those objects, such as requiring the player to use a skill received when completing the previous Destiny quest. The Destiny quests were also influenced by the 3 systems.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Omni Framework proposed new monitoring systems and quest structures for dynamic quest. While there are multiple researches on generating quests that matched the player’s playstyle, Omni Framework was unique in its ‘Destiny System’ and how it applied dynamic quest structure to that system.

MPQ-Generator used different approach in task monitoring for quests. Instead of monitoring if certain actions had been performed by the player, MPQ-Generator used a more flexible checking, that was a goal state condition check. A quest’s task was considered completed only when specific conditions were met. MPQ-Generator’s grammar-based approach meant that the generated quest cannot be modified mid-quest.

Jens van de Water's [43] framework was capable of generating quests autonomously in a dynamic game world. Here the dynamic aspect of the game world not only applied to how game objects interacted, but also applied to how quests were able to affect other quests. Completing a quest could affect another quest narrative and its contents. Doing 2 related quests in parallel could bring a player to a unique content that could not be reached if one of the quests was completed before starting the other. All these dynamic interactions and relationships between the quests were managed by the framework. The framework contained all existing quests and their current state. The framework was capable of evaluating whether existing quests, and quests that are to be generated, can be initiated and completed. The framework guaranteed that completing a quest would not result in a certain quest or part of a quest being locked out from players or impossible to complete.

MPQ-Generator did not focus on how existing quests affected newly generated quests. Thus it allowed players to possibly fail a quest.

One approach to multiple paths quest generation was introduced by Alex Stocker and Chris Alvin [44]. In their work, a quest was constructed from the combination of 'verbs' and 'nouns' components. The combination of a verb and a noun was called an 'action'. Each verb and noun are connected in hypergraph in many-to-one fashion and this relationship were used to determine how each action/node of the generated quest would be dependent on each other. The quest generation process started by generating a base quest according to the hypergraph. In this state the quest was linear and void of path/choice players could take to complete the quest. Then the quest was traversed backward to create hyperedges. Any actions/nodes that had more than one-to-one relationship could be grouped together as one hyperedge, which represented that this part of the quest 'could be completed in any order.' An example can be seen in Figure 17 (top image). Here 'Collect Rope' and 'Collect Marionberry' can be grouped together since the nouns 'rope' and 'marionberry' have relationship to the verb 'collect'. The next phase is replacing the one-to-one link with parallel path that starts and ends similar to the start and end point of the link, as seen in the bottom image of Figure 17.

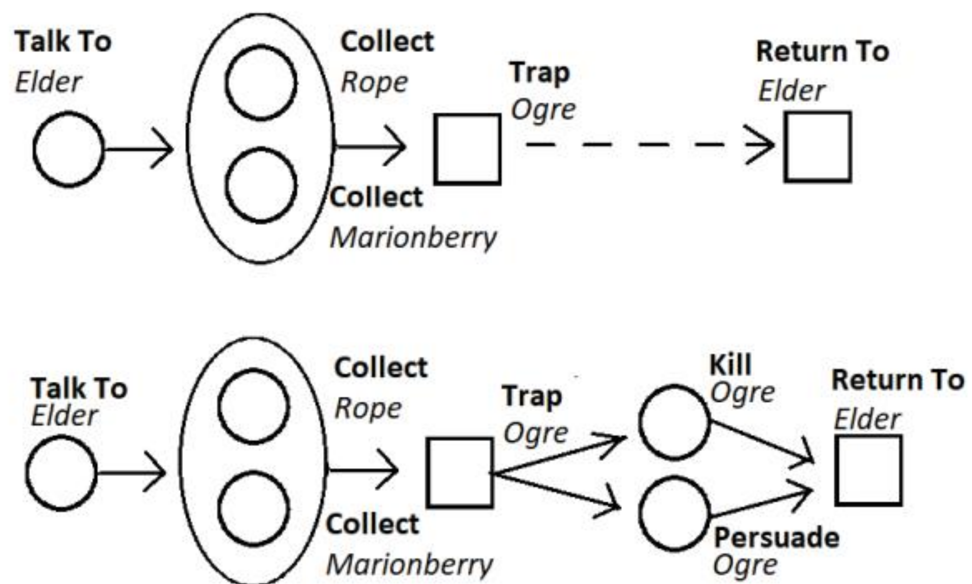


Figure. 17: (Must turn all Figure number to auto-sequence) : [Top] Base quest after collapsing some actions into hyperedge. [Bottom] After adding parallelism to the base quest. [44]

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

In Figure 17, multiple paths of the quest come from generating parallel actions that can bring the quest state to the same conclusion even with different 'actions.' The parallel actions / paths here are constructed purely using the relationship between keyword used in each state / node of the quest. This is different from MPQ-Generator approach where the paths were not determined during the quest generation, but instead obtained by running the generated quest through to see how many ways the quest could be completed. In addition, MPQ-Generator used Token system in order to assign in-game objects to each 'action' of the quest instead of using hypergraph or relationship link between the objects and the 'actions'.

Dynamic and nondeterministic quest generations mainly focused on a variety of quests and how the quests affected the game world. Not much attention was put on how players could complete the quest and how to monitor the state of the quest in these dynamic worlds. MPQ-Generator aimed to map all possible sequences of actions players could take to complete generated quests. This knowledge turned players' freedom of action and quests' flexibility into tangible measurement.

3.4 Quest Monitoring and Adaptation

There were attempts to introduce higher degree of flexible solution to quests. In table-top RPG, if players had ideas on how to resolve a quest, they were able to dynamically 'negotiate' with the human gamemaster of that game section. Human gamemaster allowed quests to be flexible and adaptive to the current situation of the game.

For example, in a 'recover stolen jewel' quest where the quest giver asked players to retrieve back a stolen accessory, players might tell the gamemaster that instead of fighting the thief, they wanted to 'haggle with the thief' (exchanging the jewel with equally value object), 'hire another thief' (to steal the jewel back), or 'fabricate the jewel' (to fool the quest giver) and so on.

On the other hand, computer RPG did not have a human gamemaster. Quests could only play out as developers had designed. Sullivan's Grail Framework [45] attempted to implement a gamemaster's flexibility and diversity into computer game quests. The Grail Framework's quest manager was adaptive to the current situation within game worlds. It monitored game world state, completed quests, active quests, and other attributes. Quests in Grail Framework used rule/condition base structure. Instead of performing certain fixed tasks to advance a quest forward, there were rule/condition that when met, would advance the quest forward or change the result of the quest. Additional independent rule/condition with similar structure could also be attached to the quest and make certain part of the quest valid or invalid (e.g. a player could not detect a hidden door without learning about it beforehand, unless the player had 'thief' profession).

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

The Grail Framework was implemented in Mismanor [46], a social interaction focus game. It was used for giving appropriate quests to players according to the current plot point and game condition. Quests in Mismanor were used to advance the plot of the game forward from one point to another until the story concluded. Quest structure consisted of “Quest Name”, “Intent” (When a quest was considered completed), “Pre-conditions” (when the quest was valid), “Starting States” (conditions that determined which ‘scene’ would be played when the quest was received), and “Completion States” (conditions that determined which ‘scene’ would be played when the quest was completed), and “tag” (associated character and location) as shown in Figure. 18. Each quest also had ‘Scenes’ attached to it. Each ‘Scene’ contained dialog and effects it had on the game world. A quest selected a scene to play out according to its state.

“

Upon receiving a quest, the appropriate scene is chosen based on the most complex state that has been matched. Each quest has a default starting scene which has no state pre-requisites.

“ *[46]*

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

The Grail Framework used game state checking to determine a direction to move a quest forward, similar to MPQ-Generator. Unlike MPQ-Generator, The Grail Framework required human authoring and input when used. MPQ-Generator also focused more on generating quests with multiple ways to complete a single objective.

Quest Name	Break up the lovers
Intent	Remove status(Violet, James, “dating”)
Pre-conditions	trust (Colonel, Player) > 40 hasKnowledge(Colonel, “Violet is dating James”) status(Violet, James, “dating”)
Starting States	1. Default – Break them up 2. friend (Colonel, Player) > 60 – Woo Violet variant 3. friend (Colonel, Player) < 20 – Woo James variant
Completion States	1. ~status(Violet, James, “dating”) 2. ~status(Violet, James, “dating”) ^ status(Violet, Player, “dating”) 3. ~status(Violet, James, “dating”) ^ status(James, Player, “dating”) 4. status(Violet, James, “eloped”) 5. friend (Colonel, James) > 50 6. ^ trust(Colonel, James) > 40
Tags	Characters: Violet, James

Figure 18: Quest structure for the example quest to break up Violet and James [46].

3.5 Quest Management and Evaluation

Analysis of ReGEN as a Graph Rewriting System for Quest Generation [47] (ReGEN) was another narrative generation system, developed by Ben Kybartas and Clark Verbrugge. ReGEN used graph-rewriting approach in its generation system. It focused on generating side-quests for role-playing games (RPGs). The game world was a directed labelled multigraph where nodes represented objects within the game and edges represented relationship between the nodes. Objects could be anything within the game world, from NPC, item, to Location and so on. A quest was a set of narrative events that could be represented by graph of ‘event node’. Here the edge between events indicated which event must happen before another event.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

When a quest was generated, the game world was examined for ‘potential story’ using ‘initial rewritten rule’ (IRR). If there existed relationships that matched the IRR, the initial quest structure and the quest that belonged to IRR were selected as the ‘initial graph/quest’ and the objects that held the relationship were selected

as part of the quest. If NPC A hated NPC B, NPC A could be a quest giver in 'murder quest' and the target is NPC B, for example. After the initial quest was generated, the quest would be rewritten again using the 'secondary rewritten rule' (SRR). The SRR examined the quest's events (Narrative Condition), seeing events that could be rewritten under appropriate conditions (Game World Conditions). Then it examined the game world for those conditions to see if the rewriting would happen or not. Figure. 19 shows an example of how SRR rewrites a single 'Murder Event,' the node within the oval highlight, into a set of 3 events with 2 branches, shown in the rectangle highlight. The SRR used here is shown in Figure. 20. This rewriting also happened in real time during gameplay, not only during the quest generation. Thus, the quest was continuously rewritten to adapt to the changing in game world.

ReGEN's quest was capable of changing its content to accommodate the changing in game world as the quest played out. MPQ-Generator used a different approach. MPQ-Generator analysed all possible ways that a quest could be completed/failed from quest generation phase, including the changing in game world condition from players' actions.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

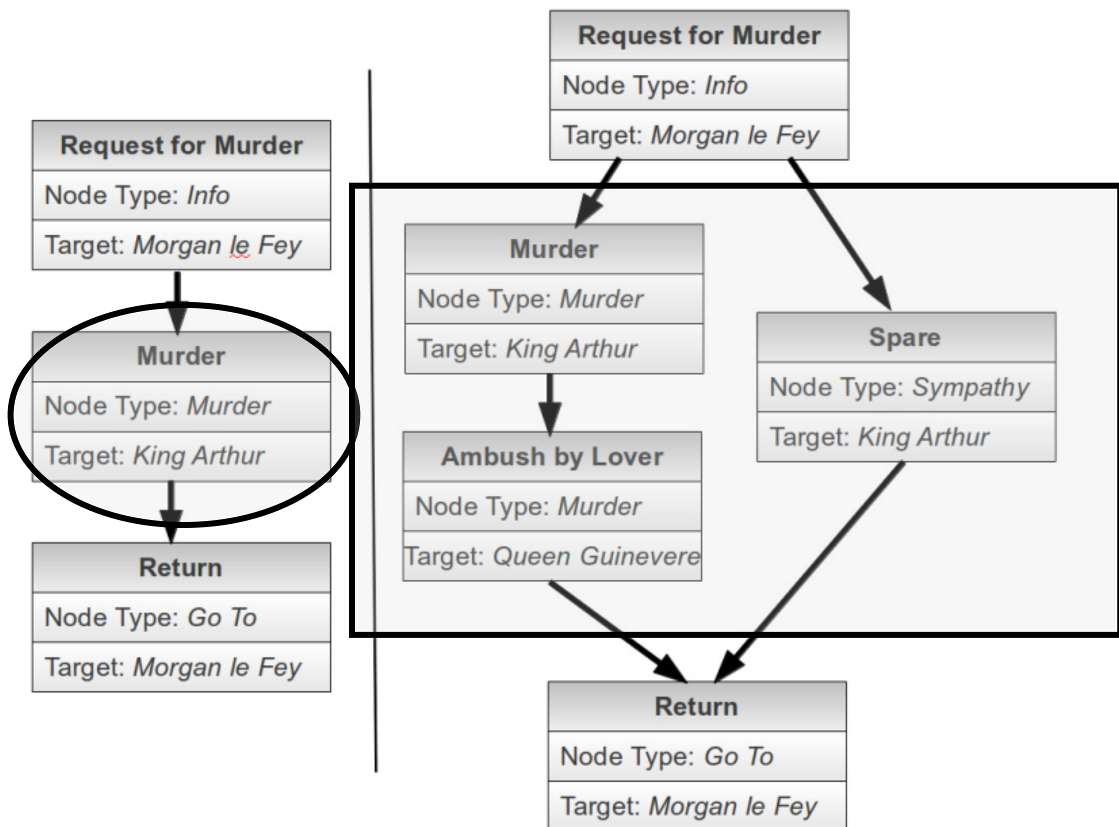


Figure. 19: Example of a generated quest. (left) The quest in IRR state. (right) The quest after SRR is applied and the quest rewritten. Taken from [47].

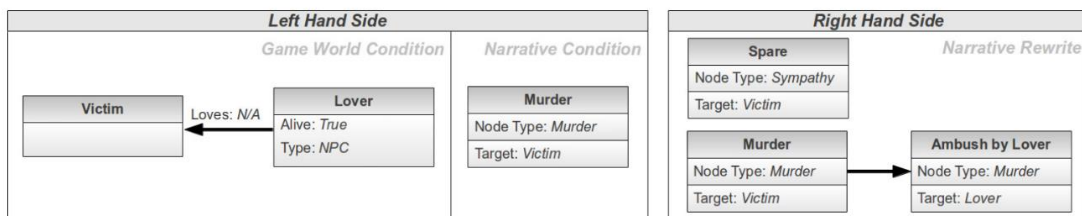


Figure. 20: Example of SRR; if all conditions of the left-hand side are true, then the quest will be rewritten as the right hand side. Taken from [47].

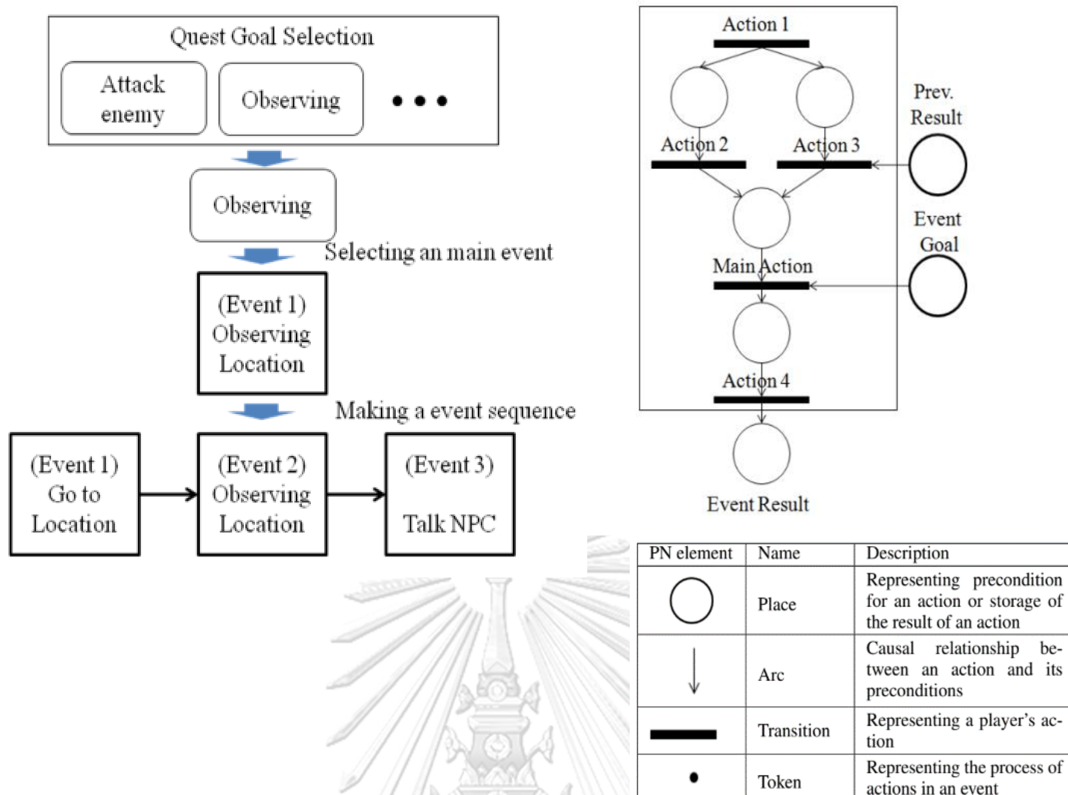
ReGEN also introduced a metric model for evaluating multiple qualities of a quest, such as length, uniqueness, and narrative richness. This metric allowed ReGEN to compare a quest that was generated by its system to a quest from other sources. It should be noted that these metrics only used tangible attribute of a quest to evaluate its quality such as number of events, gold rewards, number of branching paths, and weight of player choices. There was no objective value from human users that affected the evaluation.

This metric was modified and implemented into this thesis as a measurement tool. This gave a solid baseline to compare the quality of quests generated from this thesis with quests from other methodologies.

Lee and Cho's [37] used Petri Net model and its token structure to construct quest plots. Each state in a quest was considered as a 'GameState', a player action was regarded as a 'transition', and an 'arc' represented the relationship between action and the action preconditions. A quest was defined as a sequence of events that formed the narrative of the quest. A quest was generated by chaining multiple 'events' together. Each event contained pre-condition(s) that must be met by the previous event's post-condition if they were to be connected, as shown in the left of Figure. 21. By using Petri Net, passive element (such as conditions) and active element (such as actions) could be identified within an event and the current state of the event could be determined. Petri Net was a strong tool for modelling parallel and independent actions and conditions. Used in quest monitoring, events and conditions transition within quests and game world could be managed with certainty such that an event would only trigger when proper conditions were met (tokens were available).

Petri Net focused on quest monitoring during play, while MPQ-Generator focusd on quest monitoring during quest generation.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University



PN element	Name	Description
○	Place	Representing precondition for an action or storage of the result of an action
↓	Arc	Causal relationship between an action and its preconditions
▬	Transition	Representing a player's action
•	Token	Representing the process of actions in an event

Figure. 21: A conceptual quest generation process (left). A petri net event representation (top-right). A table of standard petri net element (bottom-right).

3.6 GameState Quest

The quest generation system CONAN by Vincent Breault, Sebastien Ouellet, and Jim Davies [48] used GameState approach to its quest structure similar to MPQ-Generator. In CONAN, each non-player-character (NPC) had a preference state they wanted to be. If they were not in such state, they would attempt to generate quest(s) that changed the current GameState (initial state) to the state they preferred (goal state). The CONAN system generated a quest, along with sequence of actions players must do, so that the goal state was reached. The CONAN system's simulation system was turn-based where NPC and player would take turn performing actions. In each cycle, NPC would generate quests using CONAN quest generation system, then players would perform an action to affect the game world. The result of player's action was then used as initial state for generating quests in the next cycle.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

While CONAN mentioned that their quests used GameState approach to check whether a quest was completed, there was no mention of the number of approaches a player could take to reach such state. MPQ-Generator focused on generating a quest along with the total number of approaches that could be taken to complete it. MPQ-Generator also considered NPC reaction against the changing in game environment and other NPCs.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

4. Methodology

4.1 Overview

Multi-Path Quest Generator (MPQ-Generator) proposed by this thesis used Structural Analysis approach from [33] as its base framework in its quest generation and quest structure along with additional rules from [39]. These works were summarised in section 3.1. The structure of game world and relationship between objects within the game world were based on ReGEN [47]. ReGEN's evaluation metrics was also used to evaluate quests that were generated by MPQ-Generator.

MPQ-Generator introduced GameState checking as a mean to monitor and manipulate quests during generation. The monitoring system allowed MPQ-Generator to determine how many ways players could tackle a quest in the current game environment. If completing the quest was not possible or the quest was too restrictive, a new quest could be generated until a completed quest was created.

The ability to measure player freedom allowed the quest generation system to generate quests with higher flexibility without compromising their integrity.

4.1.1 Quest Structure

QuestFrame

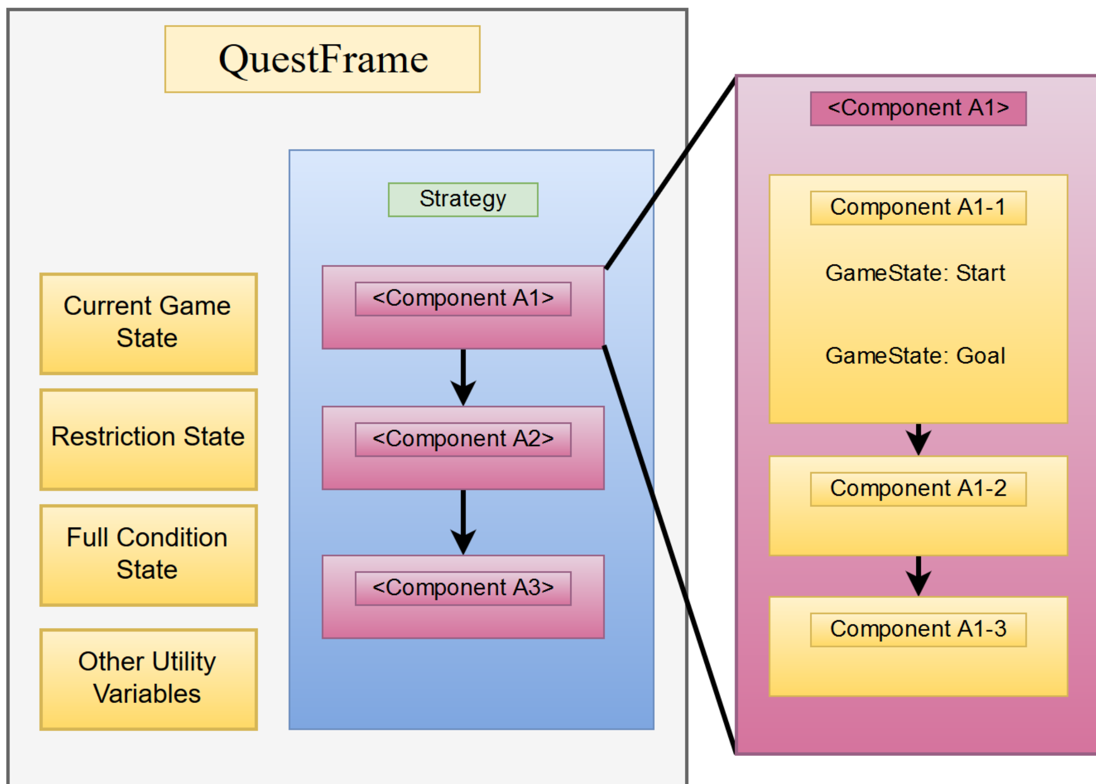
As shown in Figure. 22, a QuestFrame is the main Framework of a quest. It contains a strategy object and all other objects and functions needed for generating a complete quest. Various information regarding quest generation is also stored.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

The quest generation system reads quest information from QuestFrame when constructing a quest.

Strategy

Strategy can be described as an 'arc' of the quest. Each strategy contains a set of <Component>. The size of each strategy after breaking down all <component>, which is its number of components, is limited in order to prevent overlong quests. This strategy structure is similar to Parberry's strategy.



[Figure. 22] A portray of hierarchy and connection between elements. The arrow represents how the quest state advances forward between each point in the quest. The right <Component A1> shows how a single <Component> can be broken down into multiple Components.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

<Component>

<Component> is the smallest part of the pre-determined quest structure before the quest detail is constructed. <Component> can be broken down into either other <Component>s or finite Components that cannot be broken down. Compared to Doran and Parberry's work [33], a <Component> is equivalent to their <Rule>.

For example, <steal> can be broken down into [<goto> stealth take] or [<goto> <kill> take]. Here, 'stealth' and 'take' are the finite Components that cannot be broken down further. The process of breaking down <Component> is explained in section 4.2.

Component

A Component represents a change in game world through a player's action. This change must take place for the quest to move forward and eventually finish. Each Component contains "Start State" and "Goal State". Start State is the condition that the game world must have before the Component is selectable as a part of a quest. Goal State is the condition of the game world that must be satisfied for that part of the quest to be considered complete. After a goal state of a Component is satisfied, the game can advance to the next part of the quest.

Compared to Doran and Parberry's work [33] in term of hierarchy, a Component from this thesis is equivalent to their Atomic Action.

GameState

A GameState is an object that stores information that makes up the state of the game world. Each GameState is a collection of multiple 'GameCondition' information such as a player's health, location of each NPCs, and player reputation. GameState used in a Component can be categorised into 2 types, Start State and Goal State. Figure. 23 shows 3 components, each with different Start States and Goal States.

- **Start State** is the requirement condition that the game world must satisfy in order to add that Component into a quest.
- **Goal State** is the GameState that, when met, that section of the quest is considered completed and the quest can advance to the next component.

GameCondition

GameCondition is an individual variable. The collection of all GameConditions in the current game world will be called "GameState".

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

[Figure. 23] Three components, each having different Start State and Goal State.

Restriction State

Restriction State is a GameState that is created from analysing the list of Components used for generating quest details. It is used to prevent a conflict where a player action causes a quest to be impossible to do or causes a contradiction with the quest narrative.

For example, if a quest requires a player to deliver an item to NPC Jill, the player should not kill NPC Jill before delivering the item. Restriction State tells Prolog to not use 'kill' action to NPC Jill before the item is delivered.

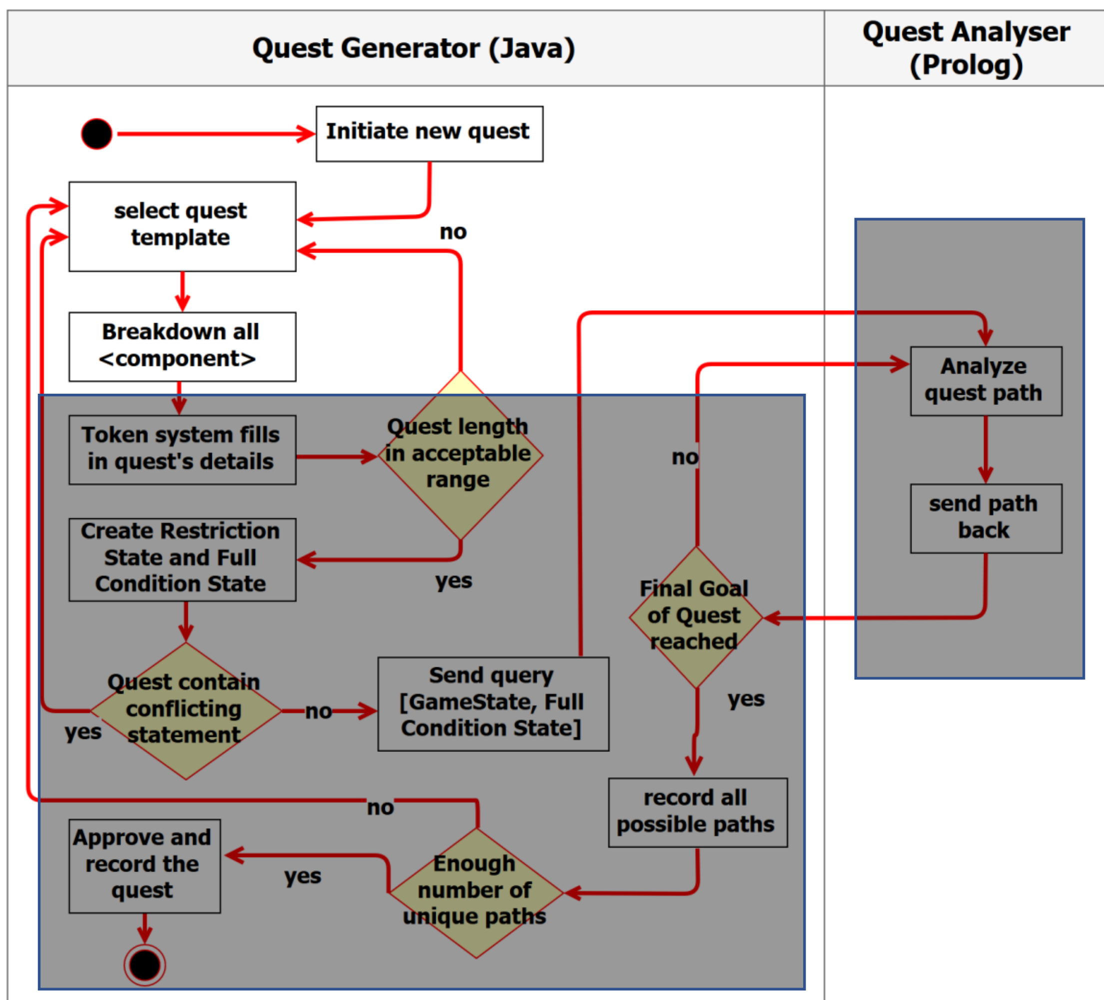
However, the Restriction State is not used to limit player's freedom. Players can still perform actions that break the quest. Restriction State is implemented only to guarantee that Prolog will not generate conflicting action path.

Full Condition State

Full Condition State is the combination of Restriction State and Component's Goal State. This is the GameState that will be used by Prolog for querying player's path of action.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

4.1.2 Proposed Method Overview:



[Figure. 24] A diagram showing the process of generating a quest from start to finish. The highlighted areas are contributions from this thesis.

CHULALONGKORN UNIVERSITY

As shown in Figure. 24, there are 2 main Components in quest generation process. Each Component manages different aspects of quest generation. The first Component is the quest generator. The process of generating quests and storing all necessary data are managed by this component.

A quest generation started by selecting a motivation and strategy (template) of the quest, then breaking down all <Components> and filling in the quest detail using objects within the game world. The processes after this step were original to MPQ-Generator. In Figure. 24, they are indicated by rectangle highlights.

If the number of Components in the quest was at acceptable range and no conflicting statement existed, the quest would be analysed by the second component.

The second Component is the quest analyser. It is a Prolog query system. Prolog was called only when the quest generator submitted its query. This Prolog query system was used when the system needed to identify if a generated quest could be completed by a player. If so, it also found out the number of paths for completing the quest.

When all possible paths were sent back, they were analysed to identify duplicate paths. If there were enough paths (as defined by the user), the quest was then approved and shown to the user.

4.2 Action Rule table Modification

Originally, Action Rule table was used to determine how Action Rule (<ACTION>) could be broken down into Atomic Action (Machado, Santos and Dias [2017] [39], which extended from Doran and Parberry [2011] [33]). Table 3 shows lists of sequence of actions from Dias' work. The initial sequence of actions from every strategy in this thesis refers to this table.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Table. 3: List of Motivation, strategy, and their corresponding sequence of actions.

Motivation	Strategy	Sequence of Actions
Knowledge	Deliver item for study	<get> <give>
	Spy	<goto> spy <report>
	Interview NPC	<goto> listen <report>
	Use item on field	<get> <goto> use *<give>*
Comfort	Obtain luxuries	<get> <give>
	Kill Pests	<goto> <defeat> <report>
Reputation	Obtain rare items	<get> <give>
	Kill enemies	<goto> <defeat> <report>
	Visit dangerous place	<goto> <report>
Serenity	Revenge, Justice	<goto> <defeat> <report>
	Capture Criminal	<goto> <capture> <report>
	Check on NPC (1)	<goto> listen <report>
	Check on NPC (2)	<goto> take <give>
	Recover lost/ stolen item	<get> <give>
	Rescue NPC	<goto> <rescue> <report>
Protection	Attack threatening entities	<goto> <defeat> <report>
	Capture Criminal	<goto> <capture> <report>
	Treat or Repair (1)	<get> <goto> use <report>
	Treat or Repair (2)	<goto> repair <report>
	Create Diversion (1)	<get> <goto> use <report>
	Create Diversion (2)	<goto> damage <report>
	Assemble fortification	<goto> repair <report>
	Guard Entity	<goto> defend <report>
	Recruit	<goto> listen <report>
Conquest	Attack enemy	<goto> <defeat> <report>
	Steal stuff	<goto> <steal> <give>
	Recruit	<goto> listen <report>
Wealth	Gather raw materials	<goto> <get> <give>
	Steal valuables for resale	<goto> <steal> <give>

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

	Make valuables for resale	<goto> repair <give>
Ability	Assemble tool for new skill	<goto> repair use
	Obtain training materials	<get> use
	Use existing tools	<goto> use
	Practice Combat	<goto> damage
	Practice skill	<goto> use
	Research a skill (1)	<get> use
	Research a skill (2)	<get> experiment
Equipment	Assemble	<goto> repair <give>
	Deliver supplies	<get> <give>
	Steal supplies	<steal> <give>
	Trade for supplies	<get> <goto> exchange

In previous works, Prolog was used to breakdown all possible paths / sequences of Atomic Action from the initial <rule>. Then a path would be chosen and delivered as a quest. The Atomic Actions within this path were considered as ‘actions’ that must be taken in sequence in order to advance and finally complete the quest. See Table 4 on how <Rule> is broken down into Atomic Action in Dias’ work [39]. The rules were chosen based on the current game world. If 2 or more rules were valid, one was chosen randomly. The broken down of <Component> in this thesis also used table 4 as a reference.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

However, the addition of GameState within Components and quest paths analysing made this property of Atomic Action obsolete. Now the new Atomic Action (Component) indicated which conditions in game world must be met for the quest to advance instead of which action players had to perform.

Table. 4: <Rule> breakdown, taken from [39].

#	**Rules**	**Explanation**
0.	<QUEST> ::= <Knowledge> | <Comfort> | <Reputation> | <Serenity> | <Protection> |	This is the root of a quest, which expands into one of the 9 motivations. Which will eventually be expanded into one of the

	<Conquest> | <Wealth> | <Ability> | <Equipment>	strategies, specific to said motivation.
1. 2.	<subquest> ::= ε <subquest> ::= <QUEST> <goto>	Go someplace. Go perform a quest and return.
3. 4. 5. 6. 7. 8. 9. 10.	<goto> ::= ε <goto> ::= goto <goto> ::= wait <goto> ::= explore <goto> ::= follow <goto> ::= stealth <goto> ::= <learn> <goto> <goto> ::= <prepare> < goto>	You are already there. Go to a known location. Wait at a location for someone or something. Just wander around and look. Follow somebody or something. Sneak by. Find out where to go to and go there. Prepare yourself before going somewhere.
11. 12. 13. 14. 15.	<learn> ::= ε <learn> ::= <goto> <subquest> listen <learn> ::= <get> read <learn> ::=<get> <give> listen <learn> ::= <goto> <subquet> examine	You already know it. Go someplace, perform a subquest, get info from NPC. Go someplace, get something, and read what is written in it. Get something, give to NPC in return for info. Go someplace, perform a subquest, examine something.
16.	<prepare> ::= <goto> <subquest>	Go someplace and perform a subquest.
17. 18. 19. 20. 21. 22.	<get> ::= ε <get> ::= <steal> <get> ::= <goto> gather <get> ::= <goto> take <get> ::= <get> <goto> exchange <get> ::= <get> <subquest>	You already have it. Steal it from somebody. Go someplace and pick something up that's lying around there. Go to someone and take something. Get something, go to someone and exchange. Get something and do a subquest.
23. 24.	<steal> ::= <goto> stealth take <steal> ::= <goto> <kill> take	Go someplace, sneak up on somebody and take something. Go someplace, kill somebody and take something.
25. 26. 27.	<capture> ::= <goto> use capture <capture> ::= <goto> damage capture <capture> ::= <goto> capture	Go someplace, use something to capture somebody. Go someplace, damage to capture somebody. Go someplace and capture somebody.
28. 29.	<defeat> ::= <goto> damage <defeat> ::= <goto> kill	Go someplace and damage somebody. Go someplace and kill someone.
30. 31.	<report> ::= ε <report> ::= <goto> report	There is nothing to report. Go someplace and report to somebody.
32. 33.	<give> ::= ε <give> ::= <goto> report	There is nothing to give. Go to somebody and give something.
34. 35. 36. 37.	<rescue> ::= free <rescue> ::= <defeat> free <rescue> ::= escort <rescue> ::= <defeat> escort	Free somebody from imprisonment. Defeat somebody, and free somebody from imprisonment. Escort somebody to someplace. Defeat somebody, and escort a different somebody to someplace.

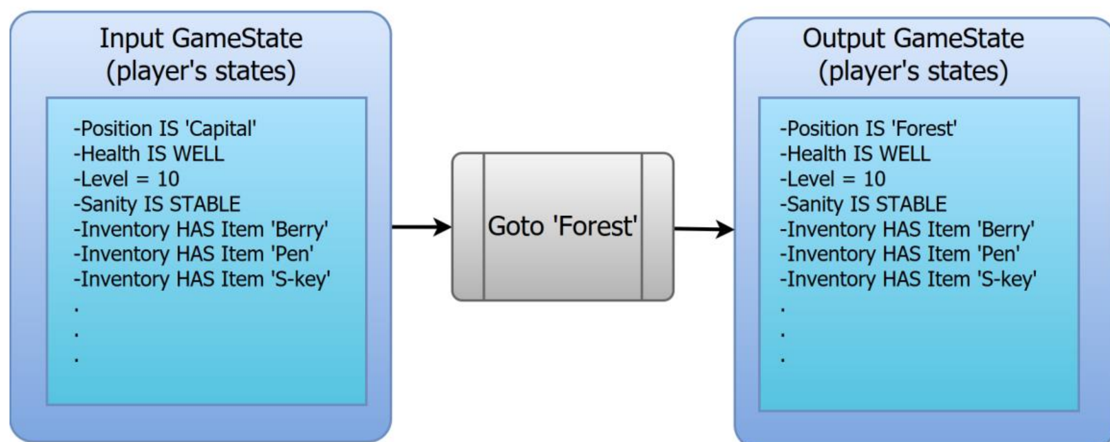
จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

4.2.1 New Action Rule table = Component Table

This thesis used Component Table, which was identical to Table 4 regarding how each <Component> could be broken down, despite the difference in content

between <Component> and <Rule>. The new Component Table defined how <Component> could be broken down into Component, similar to how the original table generated Atomic Action from <Action Rules>. The Component Table had identical number of 'rule' to the Action Rule table from Santos and Dias' work. The Action Rule table was modified by analysing change(s) in game world that took place when the Atomic Action was carried out by a player.

Shown in Figure. 25 is an example on how Action Rule was analysed. When Atomic Action [Goto 'Forest'] was chosen by a player, the player location changed from his current location to 'Forest'. Each Atomic Action had 'input GameState' and 'output GameState.' 'Input GameState' were conditions before the action happened. 'Output GameState' were conditions after the action took place.



CHULALONGKORN UNIVERSITY

[Figure. 25] When an atomic action, Goto, is executed, the player location condition is changed. Only the player information is shown in this figure.

The Start State and Goal State of a Component were created by removing all common conditions amongst Atomic Action's input and output GameStates. This was how an Atomic Action was converted into its Component counterpart.

In Figure. 25, the [Goto 'Forest'] has the following start state and goal state.

- **Start state:** "Position NOT ['Forest']" & "Exist Location ['Forest']"

- **Goal state:** “Position IS [‘Forest’]”

All Atomic Actions within Action Rule table were converted into Component and <Component> using this analysis.

Table 5 shows each Component and its complete start state and goal state. Start state described conditions that must be met for the Component to be valid during <Component> break down (see Table 2). Goal state described broadly conditions that must be met to ‘complete’ the Component. Table 5 is a modification of Atomic Actions Table from [39]. Bold words in the same row refer to the same entity.

Some of the Action Rule shared the same Goal Condition due to our simplification. Action Rule such as “Gather” and “Take” had the same converted Goal Condition because what mattered was that the player possessed the item. How the condition was met was irrelevant.

Some action-condition pairs were simplified to avoid excessive Prolog queries, such as [Explore X], or [Listen X]. Instead of requiring the player to perform the actions on a target, he/she simply needed to be at the target’s place. Any specific Action Rule only achievable with very specific player action was simplified this way. [Stealth X] could not be generalised into a single abstract state because each game implemented stealth system different from each other. Metal Gear Solid 3 used “Camouflage level” that interacted with only terrain texture, while Dishonored used shadow system which was totally different.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Likewise, instead of having dedicated dialoged action and even more ‘exchange information’ action, all “information (X)” within Table 5 were simplified and assumed to be instantly transferred or presented. This was done to improve the performance of the system. Table 6 shows how the Action Rule from the Action Rule Table were translated into GameState that must be satisfied.

Table. 5: This table outlines Components and their start state and goal state.

#	Component	Start State	Goal State
1.	**ε**	None.	None.
2.	Capture **X**	**Somebody** is there.	**They** are your prisoner.

3.	Damage **X**	Somebody or something is there.	**It** is more damaged.
4.	Defend **X**	Somebody or something is there.	Attempts to damage **it** have failed.
5.	Escort **X**	**Somebody** is there.	**They** will now accompany you.
6.	Examine **X, Y**	Somebody (Y) or something (Y) is there.	You have **information (X)** about it.
7.	Exchange **X|Y, Z**	**Somebody (Z)** is there, they have **something (X)** and you have **something (Y).**	You have **(Y)** and they have **(X).**
8.	Experiment **X, Y**	Something (Y) is there.	Perhaps you have learned **information (X)** what **it (Y)** is for.
9.	Explore **X**	**There** exist.	Wander around **there** at random.
10.	Follow **X**	Somebody or something is there.	You will now accompany **them**.
11.	Free **X**	**Somebody** is there and is prisoner.	**They** are no longer prisoner.
12.	Gather **X**	**Something** is there.	You have **it.**
13.	Give **X, Y**	**Somebody (Y)** is there, you have something **(X).**	They have **it (X)**.
14.	Goto **X**	You know **where** to go and how to get **there**.	You are **there**.
15.	Kill **X**	**Somebody** is there.	**They** are dead.
16.	Listen **X, Y**	Somebody (Y) is there.	You have some of their **information (X).**
17.	Read **X, Y**	Something (Y) is there.	You have **information (X)** from it.
18.	Repair **X**	**Something** is there.	**It** is fixed, built or resolved.
19.	Report **X, Y**	Somebody (Y) is there.	They (Y) have information (X) that you have.
20.	Spy **X, Y**	Somebody (Y) or something (Y) is there.	You have information (X) from it (Y).
21.	Stealth **X**	**Somebody** is there.	Sneak up on **them**.
22.	Take **X, Y**	Somebody (Y) is there, they have something (X).	You have **it (X)** and they don't.
23.	Use **X**	Somebody or something is there.	**It** has affected characters or environment.
24	wait	None.	Wait for something to happen.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Table. 6: This table outlines how Action Rules were converted into Goal Conditions.

Action Rule	Goal Condition
[Damage X]	(X is Alive) + (X is damaged)
[Kill X]	(X is NOT_Alive)
[Defend X], [Escort X], [Follow X]	(Player SameLocation as X) + (X is Alive)
[Explore X], [Report X,Y], [Goto X], [Listen X,Y], [Spy X] , [Stealth X]	(Player SameLocation as X)
[Free X]	(X is free)
[Capture X]	(X is captured)
[gather X], [Read X,Y], [Repair X], [Take X,Y], [Experiment X], [Use X]	(Player HAS_item X)
[Give X,Y]	(Y HAS item X)
[Exchange X|Y, Z]	(Z HAS item X) + (player HAS item Y)
[Wait], [Examine]	(NOTHING)

4.3 Methodology

This section discusses how each part of the proposed system was developed to achieve the objective of each step within Figure. 24.

4.3.1 Creating blueprint for different type of quest.

Each type of [QuestFrame] had pre-determined set of strategy and <Component>. [QuestFrame] here could be compared to the combination of “Motivation” and “Strategy”. Each [QuestFrame] was manually assembled and heavily based on Machado, Santos and Dias’ [2017] work as shown in Table 3. Collection of <Component> within “strategy” in [QuestFrame] used the equivalent collection of Action Rules from Dias’ work.

4.3.2 Quest Generation

The system was developed in Java language along with Prolog. The Java part of the program was used for generating the framework of quests. When the outline of a quest was completed, the detail of all paths and actions a player must perform in order to complete the quest would be generated by Prolog. Prolog ability to search for all possible paths and its back-tracking capability were the reasons it was

used to check all possible paths of the quest, including the feasibility of the quest. The user interacted with the program only through IDE (Eclipse was used).

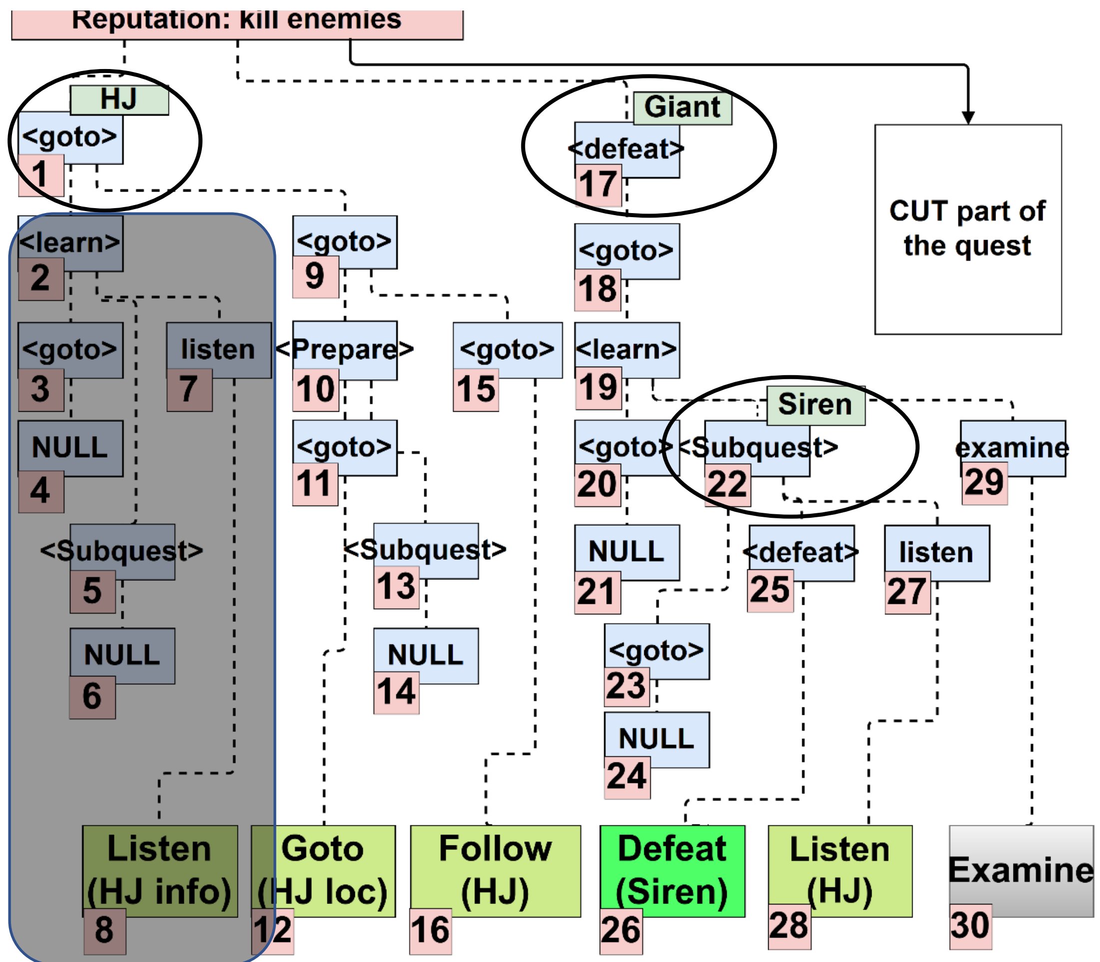
A quest generation started by initiating a [QuestFrame] object which contained all the generating quest elements. A user can determine the type of quests by modifying the [QuestFrame] setup and specifying the quest type. Then a strategy object was initiated within the [QuestFrame]. The strategy was constructed according to the quest type and settings.

Within each strategy, a list of <Component>s was created according to the selected strategy setting. The <Component>s were then broken down until no more <Component> was left. The resulting Components were stored within the corresponding strategy. The detail of Components was generated using a proposed Token system, shown in section 4.3.2. Figure. 26's shows an example on how <Component>s could be broken down.

Figure. 26's rectangle highlight shows that Component "[7] listen" was broken down from a <learn> that was generated when <goto> was broken down. The 3 oval highlights shows where and when the Tokens of this quest were generated. See section 4.3.3 for more detail on Token.

Figure. 27 shows [QuestFrame] storing the initial set of <Components> and the broken-down Components separately.

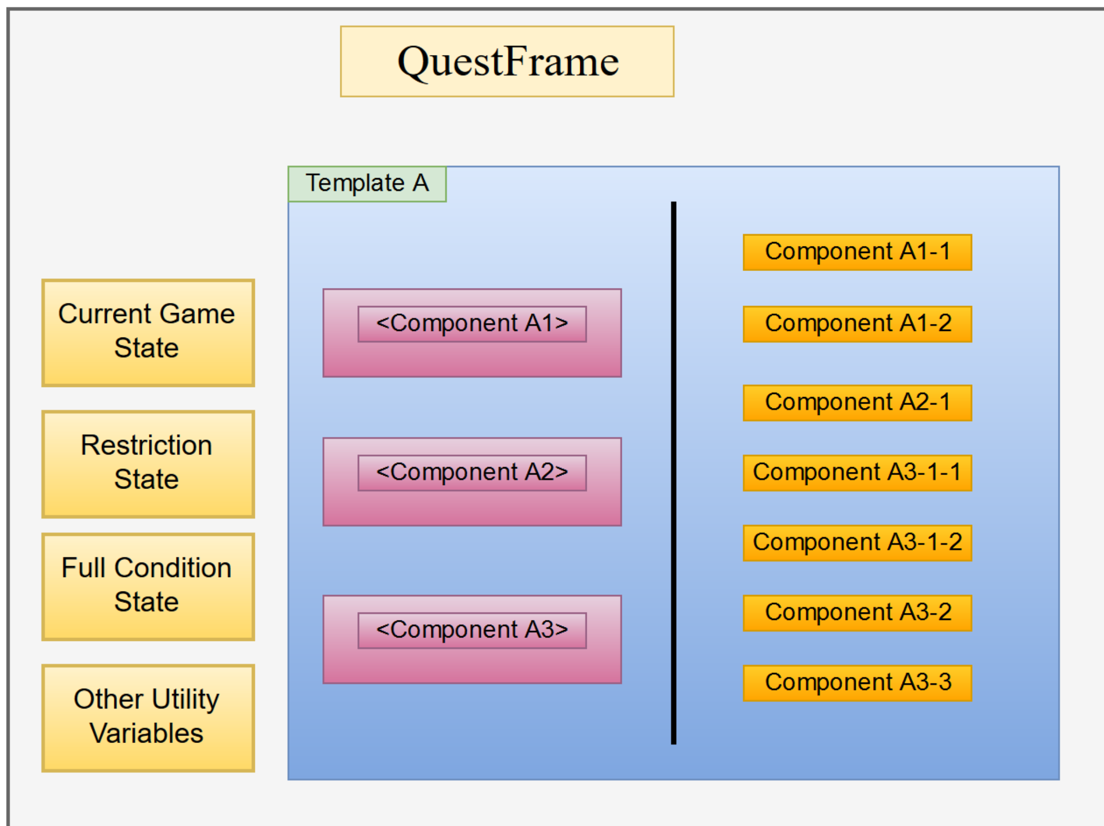
After the list of Components was created, the quest outline could be considered completed. At this point the quest size would be checked by counting the size of Components. If the quest was too short or too long, it would be discarded. A new quest would then be generated to replace it until the quest passed the required size.



[Figure. 26] How initial <Component>s get broken down into Components.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

The next step before filling in quest detail was to select a quest giver object. A quest giver could either be an NPC or an object that was designated 'quest_giver' characteristic. A generic object, such as a potion bottle, could not be a quest giver. More valuable objects, such as a map or a scroll, could be a quest giver. When a quest giver was selected, the player position would be changed to the location of the quest giver. This was to simulate the situation where the player just received the generating quest from the quest giver. Quest givers did not have to be a part of any quest.



[Figure. 27] Status of a quest after <Component>s are broken down. The resulted Components are kept separated and do not replace the initial <Component>s.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Each quest giver also possessed a "quest level" property. When a quest giver was selected, its quest level would be read. The system would not pick up objects with higher quest level than the quest giver. This was to avoid situations such as a thief (level 2) giving an objective to kill a king (level 3). However, this rule was not absolute. If there was no available mini-Token with the right criteria during mini-Token assignment, an object with higher quest level could be selected provided that the Component allowed the object to be randomly selected from the game world.

4.3.3 Token

Our Token system retrieved objects from the game world and assigned them in the generated quest. The system made the objective of each Component more consistent and related to one another. A new Token was created when a specific

<Component> was broken down into a specific Component or <Component>. A Token contained 3 mini-Tokens. Each mini-Token represented different data related to the in-game object that the Token represented. The in-game object was selected randomly from all objects that had lower or equal quest level to the quest giver. Then a Component (starting from the first Component where the quest began, up to the final Component that ended the quest) was assigned a mini-Token. Components might accept only certain types of mini-Token. The 3 mini-Tokens were:

1. [Information (level 1)] (or info)
2. [Location (level 2)] (or loc)
3. [Itself/Core (level 3)] (or the object name)

Each mini-Token could be assigned to a Component only once. Some Components accepted only certain level mini-Token and/or mini-Token from certain types of in-game objects. If a mini-Token was assigned to a Component, any lower level mini-Tokens became unavailable and could not be assigned to another Component. Some Components required two or more mini-Tokens to be assigned.

In Figure. 28, the quest "The Lord of Undvik" from the game "The Witcher 3" is represented using <Component>s prefix traversal quest tree according to the breakdown rule. The non-null leaf nodes are the Components of the quest. The banners just above the <Component>s are the generated Tokens. These Tokens were generated at every starting node of the quest and then at every <subquest> node. However, if all children nodes of the Token node were NULL, no Token would be generated. For Token assignment, each Component tried to use the closest existing Token in the tree (only Tokens with lower node number than the Components' were available). As seen in Figure 26, the Component number 8, 12, and 16 (yellow highlight) used mini-tokens from the <Component> number 1. For Component number 26, it used [Siren] mini-token instead of [Giant] because the [Siren]'s <Component> was 22, which was closer to 26 than the [Giant]'s <Component>, which is 17.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

A “Listen [28]” Component (see Figure. 26) only accepted core mini-Token and the Token’s object must be a character. The token [Siren] and [Giant] were not used because those ‘objects’ were ‘monster’ and not ‘character’, making them illegal, even though they were closer than the token HJ. It was possible that the “Listen” Component did not have available Token to assign. The system fixed this problem by randomly selecting a related object, reusing the used mini-Token that matched the requirement, or discarding the whole quest and generated a new quest, depending on the specification of the Component. In Figure. 26 “Listen [28]” Component allows a used Token to be used again, thus HJ gets assigned as its mini-Token.

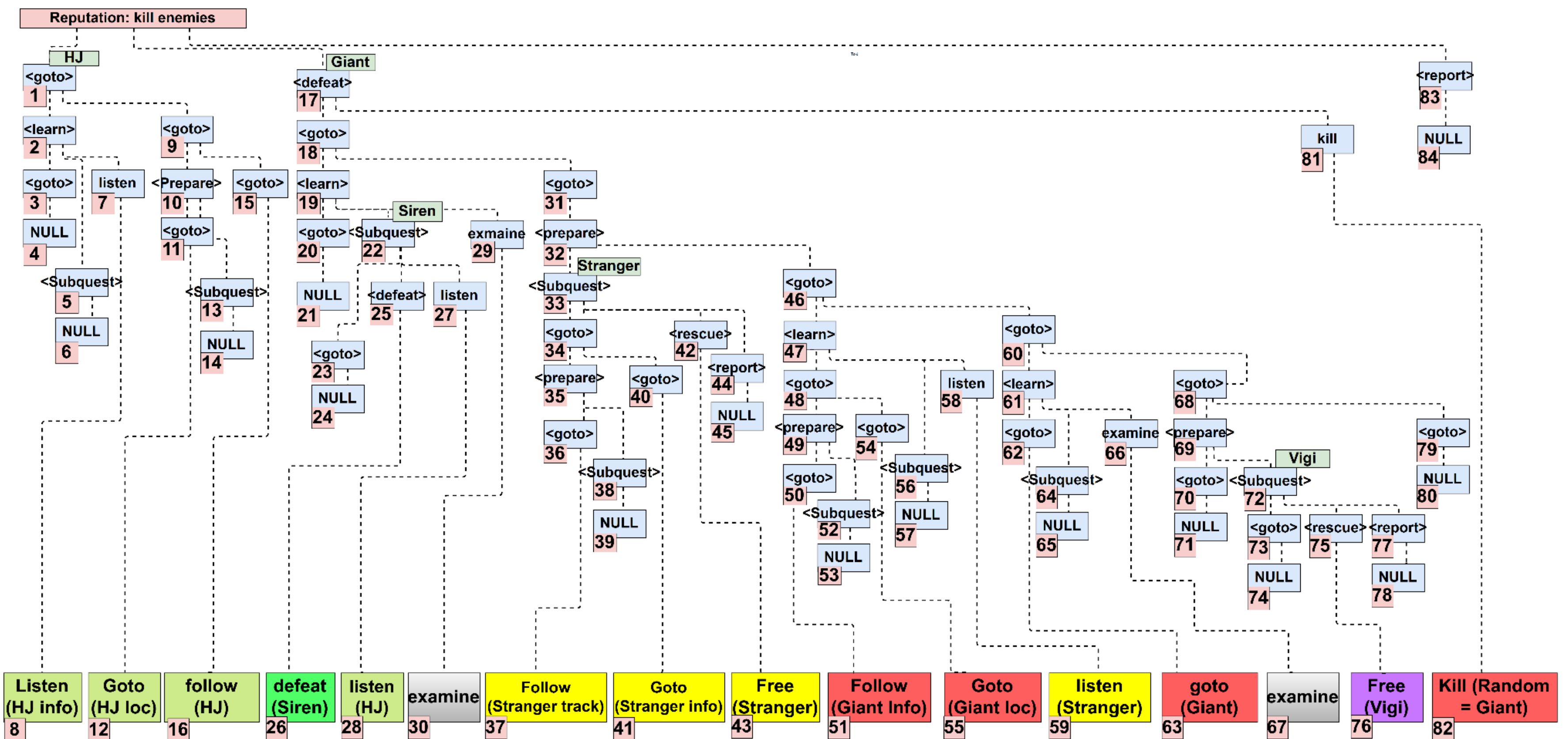
Some Components such as “Examine” (see Figure. 26, black highlight) was not assigned mini-Token because it did not need an associated object.

When an Information mini-Token was assigned to a Component, that Component’s objective would be about the action of seeking knowledge regarding the mini-Token’s object. Likewise, Location mini-Token represented the location related to the object. Core mini-Token represented the object itself. Components with core mini-Token mainly demanded players to perform some task directly on the object.

Some combination of mini-Tokens and Components might result in the same goal condition. Components were assigned mini-Token in order according to its node number. The node number was assigned according to prefix traversal of the quest tree.

Figure 28 shows how the quest ‘The Lord of Undvik’ from the Witcher 3 (game) could be represented using the token system. The result closely resembled the official quest content and objective.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University



[Figure. 28] Status of a quest after <Component>s are broken-down. The resulted Components are kept separately and do not replace the initial <Component>.

The next step after breaking down all the <Component>s was to fill in detail, narrative, and specific condition of the quest for it to be playable. Since Action Rule table no longer represented direct tasks that the player must perform, if the system was to determine possible actions a player needed to perform to complete his quest, Prolog must be queried to get that information. Prolog then exhaustively discovered all possible paths the player could carry out to complete the quest.

However, the current quest outline still lacks necessary information required for a thorough query. Such information was contained in Full Condition State, which was constructed using Restriction State. Restriction State was used to prevent Prolog from creating conflicting sequence of actions.

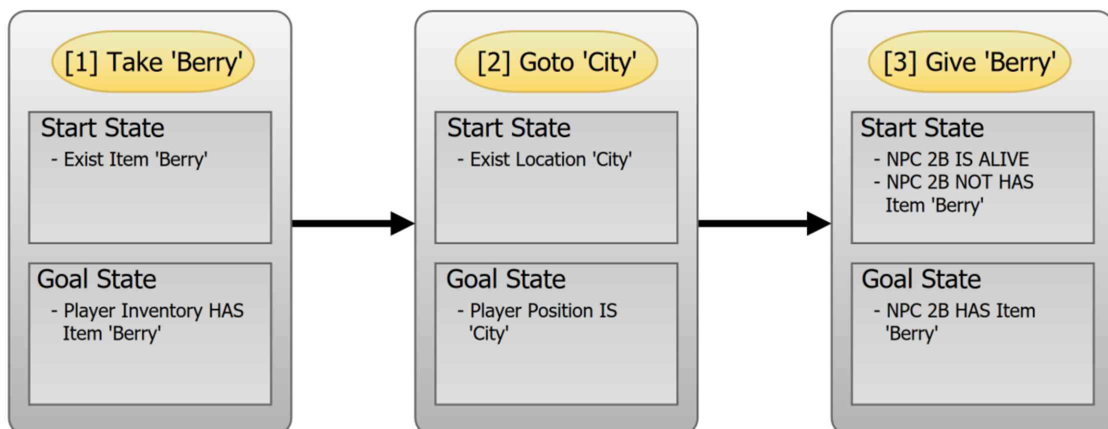
4.3.4 Obtaining Restriction State

When finding the actions that satisfy the current task requirement, Prolog only sees the 'current' task's goal state. This may result in Prolog selecting an action that makes later tasks' goal states unreachable. When this happens, Prolog has to backtrack, resulting in a longer quest path finding process. In order to avoid this situation, all information regarding all tasks within a quest must be provided. This is where a Restriction State comes in.

To obtain a Restriction State for a quest, all Components within the quest had their StartState and GoalState broken down into a list of GameStates from all Components. Then the system started to read the list from back to front, while assembling the necessary GameState of which conditions must be met if the last GameState was to be achievable from the first GameState. The 3 level Components, [[1]"Get Berry" >>> [2]"Goto 'City' " >>> [3]"Give Berry"], are used as an example. See Figure. 29.

To construct a Restriction State, first, the 3rd level component, [Give Berry], was analysed. It required the player to [deliver 1 unit of Berry to NPC 2B]. The Component was pre-determined on how it affects the Input GameState. Thus the 'Effect to input GameState' of [Give Berry] Component in this example was "NPC 2B inventory has 1 additional unit of Berry" and "player has 1 fewer unit of Berry". This can be seen in Figure. 30 within 'Effect to Input GameState'.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University



[Figure. 29] Three components, each having different Start State and Goal State.

However, the 'effect' was not yet usable. The 'effect' had to be generalised and then negated, forming a 'Resolved Conditions'. The Resolved Condition would then be used to create Restriction State. Effect such as "NPC 2B Inventory Item 'Berry' INCREASE 1" were generalised into "NPC 2B Inventory HAS Item 'Berry' " and finally negated into "NPC 2B Inventory NOT HAS Item 'Berry' ", as shown in Figure. 30 's Resolved Conditions. The generalisation was done in order to create consistency and allow the system to focus on the essential GameConditions. The negation was done in order to prevent the condition from being completed before the Component for creating that condition was actually reached.

The 'Restriction State' was the result of [current GameState] + [generalised & negation of 'effect'].

CHULALONGKORN UNIVERSITY

Each Component had its associated Resolved Conditions template created beforehand and referred to when creating its Restriction State. Objects (Token) within the Component would fill in the detail of the Resolved Conditions template.

Then the system created Restriction State from the Resolved Conditions and the Start State of the 'component'. This was done by combining GameConditions from both states. Duplicates were ignored. Then a level "[X]" indicator was attached to each GameCondition depending on the current level of Component (X = 3 in this example).

Figure. 30 (bottom left) shows the Restriction State after analysing the 3rd Component [Give Berry]. The [3] in front of each GameCondition identifies the Component level that the GameConditions were created from.

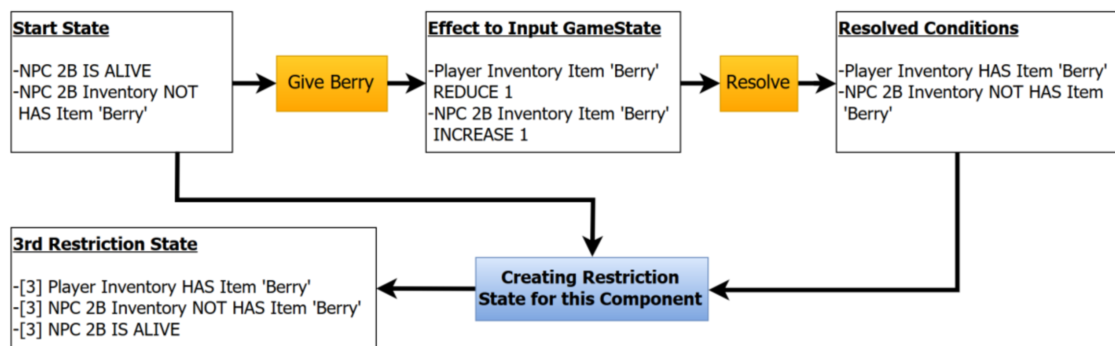


Figure. 30: 3rd (3rd level Components) Restriction State after [Give] Component is analysed.

From Figure 30, it can be seen that Resolved Conditions and Start State both has a GameCondition "-NPC 2B Inventory NOT HAS Item 'berry'". But in the 3rd Restriction State, there is only one "-[3] NPC 2B Inventory NOT HAS Item 'berry.'" This is because GameCondition within Start State has higher priority and will override any conflicting/similar GameCondition from Resolved Conditions. It may not be obvious in this example because "-NPC 2B Inventory NOT HAS Item 'berry'" exists in both places, but what happens here is that the GameCondition from Start State overrides the GameCondition from Resolved Conditions. See Table 7 for a rule table that shows how each type of GameConditions are subjected to the override rule. Bold word highlights keywords used for checking whether 2 GameConditions belong in the same category and therefore can override one another. Then if the object (shown in italic in Table 7) also matches, the GameCondition in Start State overrides the corresponding GameCondition in Resolved Conditions.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Table. 7: Rule table on how GameCondition overrides another GameCondition.

BASE	OVERRIDE	RESULT	NOTE
Boolean Variable			
HAS Item *'X'*	NOT HAS Item *'X'*	NOT HAS *'X'*	*Any GameCondition with HAS, IS keyword belong in this category.
NOT HAS *'X'*	HAS *'X'*	HAS *'X'*	
IS ALIVE	IS NOT ALIVE	IS NOT ALIVE	
IS IN FACTION	IS NOT IN FACTION	IS NOT IN FACTION	
EXIST *'X'*	NOT EXIST *'X'*	NOT EXIST *'X'*	
Relationship			
BE FRIEND	BE NEUTRAL	BE NEUTRAL	
	BE ENEMY	BE ENEMY	
Location			
AT *'Y'*	AT *'Z'*	AT *'Z'*	

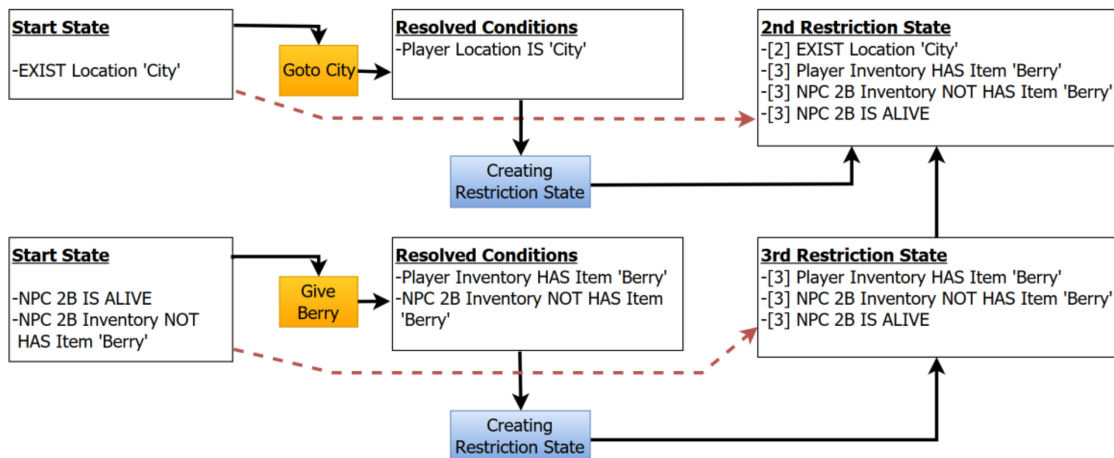
From Figure. 29, the next Component ([Goto 'City']) was then analysed and its own Restriction State at level 2 (2nd Component [Goto]) was constructed. At this point, the Restriction State did not only receive GameConditions from Start State & Resolved Conditions, but also from the Restriction State of the previous Component, which contain:

- "-[3] Player Inventory HAS Item 'berry' "
- "-[3] NPC 2B Inventory NOT HAS Item 'berry' "
- "-[3] NPC 2B IS ALIVE "

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

These GameConditions were passed to the 2nd level Restriction State as shown in Figure. 31. However, Component [Goto] was special. It did not contribute to the Restriction State. This was because NOT exempting [Goto] would result in multiple locations that player could not reside to initiate the quest. Restricting locations might create conflicts in a quest that required a player to go back and forth to the same location multiple times.

That was why the 2nd Restriction State only included "-[2] EXIST Location 'City'", and not "-[2] Player Location IS 'City'"

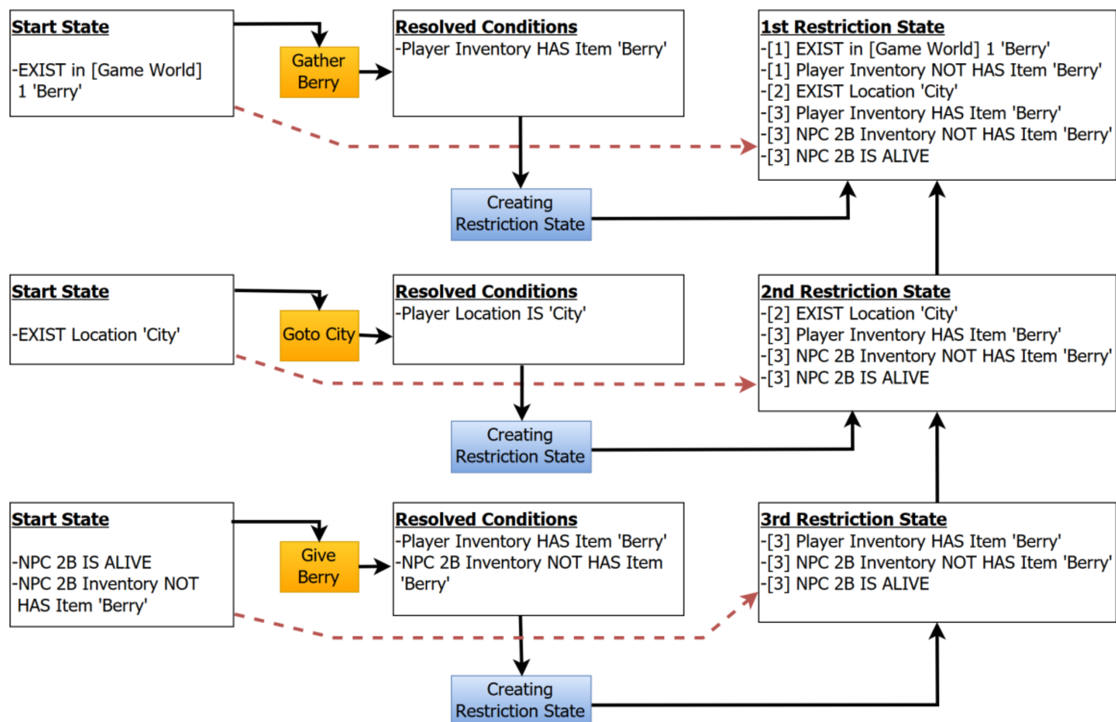


[Figure 31] 2nd (2nd level Components) Restriction State after [Goto] Component is analysed.

Finally, the last Component (or the 1st in term of quest structure) was analysed. [Gather] would give GameCondition “Player Inventory NOT HAS Berry” to the passed Restriction State. However, there was already GameCondition “Player Inventory HAS Berry” in the list. In this case, the newest GameCondition (“Player Inventory NOT HAS Berry”) would be added, not overwritten, since the conditions were from different Component level ([3] and [1]), as shown in Figure. 32.

When the process finished operating on all Components, the final Restriction State would be used to create Full Condition State.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY



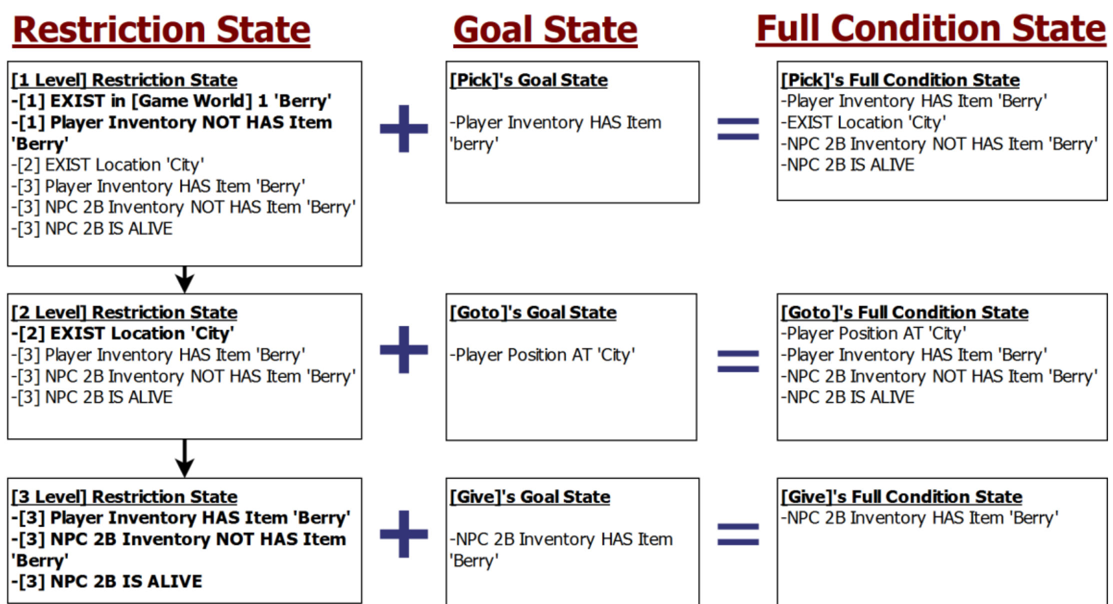
[Figure. 32] 1st (1st level Components) Restriction State after all Components were analysed.

4.3.5 Obtaining Full Condition State

After the Restriction States were created, the system was able to determine which condition was prohibited during the 'path' generation. This was implemented by merging the list of Restriction States with the Goal State of the Components. The [X] tag in front of a GameCondition in the Restriction State determined which Components it was to merge with.

If the current Component level was higher than or equal to the [X] of Restriction State's GameConditions, all those GameConditions were omitted. Continuing from the example in Figure. 32, the 1st Component [Pick]'s GameCondition "Player Inventory HAS Item 'berry'" were fused with "[2] EXIST Location 'City'", "[3] NPC 2B Inventory NOT HAS Item 'berry'", and "[3] NPC 2B IS ALIVE." This resulted in Full Condition State of [Pick].

The 2nd Component [Goto] and 3rd Component [Give] were fused with [2nd Level] Restriction State and [3rd level] respectively as shown in Figure. 33. Each combined state was called a "Full Condition State".



[Figure. 33] The process of fusing Restriction States with original components.

This meant that after the 2nd Component goal state was reached, it was no longer necessary whether 'City' exists or not. And if there was to be 4th Component, the condition that NPC 2B had to be alive would not be applied to the Starting GameState of the 4th Component. The Full Condition State was used in placed of Goal State for its corresponding Component. This was to guarantee that when the Component was completed, the GameState would not lead to any dead end situation.

CHULALONGKORN UNIVERSITY

4.3.6 Checking for impossible quest

This step used the Full Condition State to check whether the quest was possible or was 'appropriate to narrative' or not. This step checked for situations such as "NPC 2B is tasked to die at level 2 Component, but also need to be alive at level 3 Component." and discarded the quest if such situations were found.

This was done by reading all GameConditions within the Full Condition State and checking for certain conflicts. Such conflicts were identified as follows:

1. There are “IS ALIVE” and “IS NOT ALIVE” GameConditions of identical entity in Component of any level.
2. There are [EXIST ‘X’] and [NOT EXIST ‘X’] GameConditions in the Components of the same level.

If any of these rules were true, the quest would be discarded, and a new quest would be generated in its place.

Full Condition State was then used to prevent the Prolog system from choosing conflicting action between each GameState. When Prolog started to generate path from the root GameState (start State of the first Component) to the goal state of the last Component (quest end), it checked if the GameState Conditions generated by these paths conflicted with the provided Full Condition State.

By doing this, the system was able to prevent conflicting and inconsistent sequence of actions. The result path from each GameState to its next GameState retained the consistency of the original Action Rule Table, while allowing the new path generation system to perform without any hindrance from backtracking.

In the next step, the Quest Analyser (written in Prolog) would be queried to get all possible paths of action that the player could perform. The following information would be passed to the Quest Analyser:

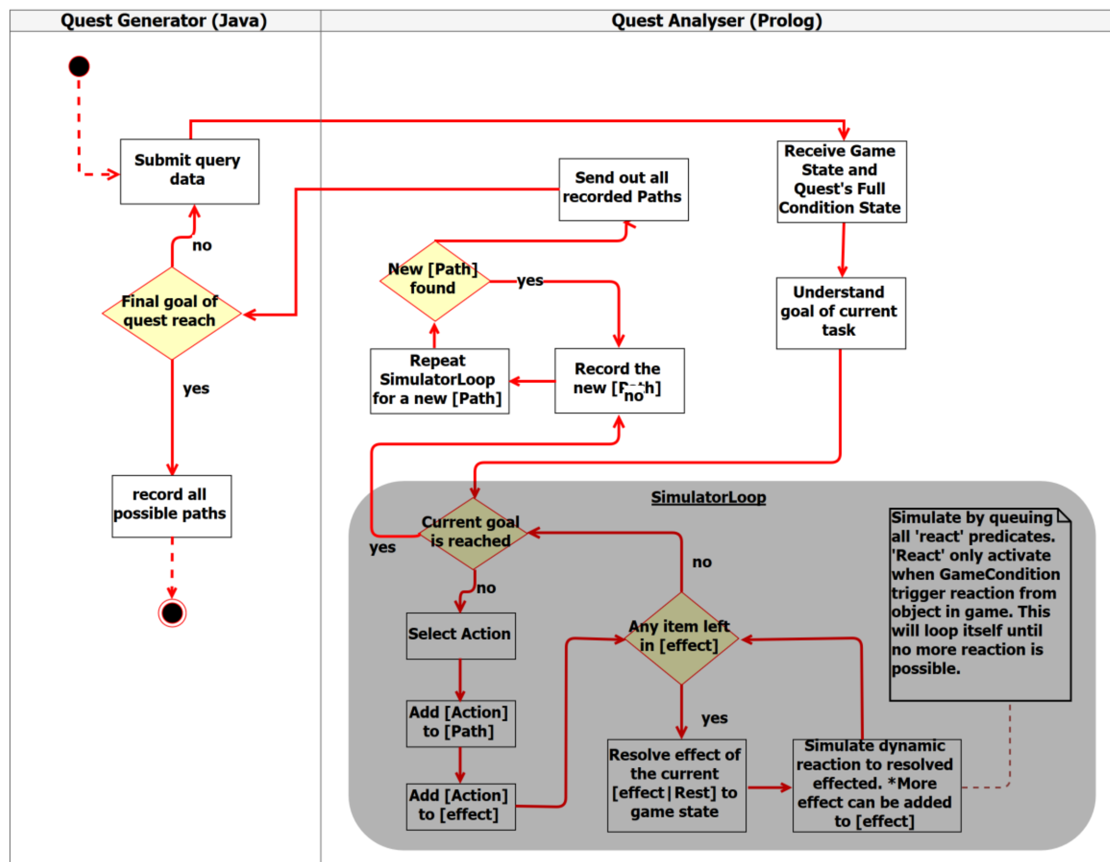
จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

- **Current GameState**: All current game conditions that are stored within the system.
- **Full Condition State:** The condition that must be true.
- (If exist) **Previous Path**: List of action the player did previously.

4.3.7 Path Finder / Quest Analysing

The Quest Analyser discovered and recorded paths to complete the quest. Figure. 34 shows a diagram illustrating the overview of this process. After the quest was generated, the first Component and its Full Condition State were sent to the Quest Analyser (Prolog) along with current GameState. The system did not send the

whole quest because querying the quest Component-by-Component made management and debugging easier.



[Figure. 34] A Diagram illustrating how all paths to finish the input quest are discovered and recorded. The highlighted area is the part where Prolog 'simulates' a player's action and game world's dynamic reaction to the action.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

From starting point, the predicate received all the mentioned data and try to discover all possible paths from the start of the Component to its goal state (Full Condition State). Figure. 35 shows a part of Prolog code queried by Java as the first predicate. In line 48-53, all of the previous information regarding the previous query attempt were retracted. Then in line 54, each individual Goal Conditions within [GA] was categorized into either character oriented goal [GC] or location oriented goal [GL]. This was because GC and GL are of different string-list length and predicate could only take input of specific length. Line 55 would forward the queried to another

predicate called "startQuestPath". This predicate checked whether the Goal Conditions were already completed (and thus needed no further action) or not. If not, the query was then forwarded to "questPathMainLoop" shown in Figure. 36.

```
45% GA here = all Goal condition that isn't seperated yet.
46startQuestPath(GA,AC,AR,LA,P,PF)
47:-
48    retractall(pathExist(ANYTHING)),
49    retractall(conditionExist(ANYTHING)),
50    retractall(counter(ANYTHING)),
51    retractall(pathExist_ac(ANYTHING)),
52    assert(counter(0)),
53    \+pathExist(ANYTHING),
54    seperateGoalType(AC,GA,GC,GL),!,
55    startQuestPath(GC,GL,AC,AR,LA,P,PF).
```

[Figure. 35] The starting query that is invoked first from Java side

```
88%Main loop to check if after 1 action and follow by resolve, the quest
89%goal is reached yet or not.
90questPathMainLoop(GC,GL,AC,AR,LA,P,LC,PF)
91:-
92    (   questPathMainLoop_Done(GC,GL,AC,AR,LA,P,LC,PF) -> writeToFile(GC,GL,AC,AR,LA,P,LC,PF)
93    ;   questPathMainLoop_Continue(GC,GL,AC,AR,LA,P,LC,PF)
94    ).
```

[Figure. 36] The main query loop

The predicate "questPathMainLoop", was the main predicate that all other loops and predicates came back to execute when they finished their queries. This questPathMainLoop was responsible for the tasks "Current goal is reached" shown in Figure. 34. The predicate "questPathMainLoop_Done" checked if all Goal Conditions were met. If true, the path would be written down using the "writeToFile" predicate. Otherwise, the query continued.

Table. 8: This table shows all possible actions.

ACTION	DESCRIPTION
Goto	Move the player character.
Direct_Attack	Attack a character.
Hire_to_Attack	Hire an NPC to attack another NPC.
Pickup_Ground	Pick up a specified item at the location.
poisoned	Poison the target NPC.
Pickup_Body	Pick up an item from dead body.
Give	Move a specified item from player to NPC.
Capture	Give captured status to a specified NPC.
Free	Remove captured status from a specified NPC.
Damage item	Give damaged status to a specified item.
Fix	Remove damaged status from a specified item.
/Bribe_add_crime	Give criminal status to a specified target.
Bribe_remove_crime	Remove criminal status from a specified target.

Table. 9: This table shows a list of reactions NPC performs when conditions are met.

REACTION	DESCRIPTION
Friend cure poison	NPC with antidote or heal ability will remove poisoned status from a friend at the same location. The NPC must have 'heal' ability.
soldier capture criminal	Soldier applies captured status to NPC with crime status at the same place.
soldier kill NPC criminal who resist	If the NPC with crime has higher level than 15, it will resist capture and gets killed.
Doctor cure poisoned	Doctor removes poisoned status from NPC at same place.
Doctor cure damaged	Doctor removes damaged status from NPC at same place.
Soldier bring captured	NPC with criminal and captured status at the same place

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

criminal to jail	with a soldier NPC is moved to jail along with that soldier NPC.
Jailbreak	Captured status is removed from NPC (in jail) with a lockpick.
drink antidote	If NPC has antidote and is poisoned, poisoned status is removed.
character die from poisoned	If NPC has poisoned status, it dies (Checked last, after all reaction fail).

Once the questPathMainLoop forwarded the query to the "Select Action" (see Figure. 34), an action predicate was selected. Table 8 shows all possible actions a player could performed. When an action was chosen, its ['action name' + 'subject of the action'] would be added to a list called [Path] and [effect]. A [Path] was defined as a list of 'action' that led to the current GameState. When the final goal of the quest was reached, the current [Path] would be recorded and sent back to the Quest Generator (See Figure. 24). An [effect] was defined as a pending list of effects yet to be applied to the current GameState.

To apply an action's effect to the current GameState, another loop of query must be used. First the [effect] of the actions would be separated into [next_effect | RemainingEffects]. 'next_effect' was the effect that would be applied to the current GameState, 'RemainingEffects' was a list containing all other effects waiting to be applied. When the effect was applied, the GameState changed. The change invoked 'reacting' predicates (Figure. 37) that served as 'dynamic reaction' simulation to the effect. All reactions are listed in Table 9. They were a set of predicates that only triggered when certain conditions were met. In Figure. 37, when the 'reacting' predicate was called, it would call "getCharList" predicate to generate a list of all characters to be used in the actual loop ("reacting_Pair"). This must be done to prevent Prolog from assigning the same object into the variables when there were more than 1 variable. One of the "reacting_Pair" predicate is shown in Figure. 38. It checked for conditions when an NPC had 'poisoned' status. If true, the predicate

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

then checked if there was any character that was the NPC's friend and was capable of healing the poisoned status.

Everytime a 'reacting' predicate evaluated to true, it would call itself again and queried to the next 'reacting Pair' predicate using the new condition (created from the previous reaction that took place). When no more 'reacting' predicate was triggered, the system checked if the goal state of the current task (Component) was reached. If not, a next action was selected, and the loop continued. Otherwise, the system recorded the current [path] (list of taken actions) and GameState. Then it checked for another possible path. If no more path exists, the system sent recorded paths and GameStates to the Quest Generator.

```
918 reacting(AC,AR,LA,P,LC,ACRS,ARRS,LARS,PRS,LCRS) :-
919%    getCharPairINI(AC,Pair_list),
920     getCharList(AC,Single_list),
921     reacting_Pair(Single_list,AC,AR,LA,P,LC,ACRS,ARRS,LARS,PRS,LCRS).
```

[Figure. 37] The reaction loop.

```
961 %Friend cure Friend poisoned
962 reacting_Pair([A|T],AC,AR,LA,P,LC,ACRS,ARRS,LARS,PRS,LCRS)
963 :-
964
965     existCharListStatusFromList(AC,A,poisoned), %is poisoned?
966
967     existRelationship(AR,friend,A,B),
968     existCharListSkillFromList(AC,B,heal), %can char2 cure poisoned?
969
970     getCharIsAliveFromList(AC,A,true), %The character must be alive
971     getCharIsAliveFromList(AC,B,true), %The character must be alive
972
973
974     delete(AC,[A,listStatus,poisoned,z,zz,zzz],ACRE),
975
976     append(P,[friend_heal_poisoned],P2),
977     append(P2,[A],P3),
978     append(P3,[B],PRE),
979
980     ACRS = ACRE,
981     ARRS = AR,
982     LARS = LA,
983     PRS = PRE,
984     LCRS = LC.
```

[Figure. 38] An example of reaction predicate, where A = char1, and B = char2.

4.4.8 Avoiding infinite loop

In order to avoid any situation where the system would run infinite number of possible paths and loop forever, some constraints were implemented.

First, the size of each [path] (also called “depth”) was limited. When the limit was reached, the system would consider that path a dead end and backtracked to find another path. This path size could be configured by users. The depth configuration used in this thesis was 3, meaning a player character was forced to perform at most 3 actions. This number was chosen because of performance issue. With this setting, each quest took around 30 min to 4 hours to generate. At depth 4, the time required became more than 6 hours at the minimum. At depth 5, the system regularly crashed due to ‘out of memory’ error. However, even at depth equal to 3, some quest strategies were still not feasible. For example, no quest was ever created from the strategy “Trade for supplies” because the quest generation was only able to find paths with depth more than 3.

Second, pattern identifier was used to discard repeated [path] and other undesirable [path]. A path found by Prolog was subjected to the following criteria before it would be viable:

1. The player character must be alive.
2. Its action, target of action, and its reaction record must not be a subset of any existing path.
3. A player’s sequence of actions must not be a subset of any existing path’s player’s sequence of actions. This excluded the target of each action.
4. The front half of the path must not be a subset of any existing path.
5. The back half of the path must not be a subset of any existing path.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Without the criteria, a simple “Goto” Component might have up to 45000 paths (mostly identical, with different ordering of actions). If only the first and the second criteria were applied, the Component still had more than 4000 paths (each one still very similar to others).

The 5 criteria included paths from all previous Components in its review. The initial setting only checked one Component at a time. However, this resulted in over

100 paths for a Component. These paths were unique, but the generation time was too large to generate a quest within a day. Thus, the setting was changed to include paths from previous Components to lower the path numbers.

After all 5 criteria were applied and previous Components' paths were included, the "Goto" Component path number came down to around 6-20 paths.

However, applying all 5 criteria denied some valid paths to be recorded, mainly short paths that could complete the specific Component within 1 or 2 actions. Due to how Prolog used depth-first-search, it was highly likely that the first [path] found was a 3-actions path. From our experiments, even though 1 or 2 actions within the [path] were not needed (not doing the action would still result in the goal state), they would still be recorded. When a [path] that only used 1 or 2 necessary action was later found, it would not be recorded because there already existed a 3-action [path] which included those actions as its sub-path.

Figure. 39 shows a path example from a quest generated from "Kill enemies" strategy. The top part of the figure shows the Full State Conditions of each Component within the quest. The bottom part shows the actions performed by the player and reactions of the actions. The left side describes the action or reaction taken, and the right side describes the subject, object, and/or actor.

After receiving the information, the Quest Generator then stored all the information for the next predicate. Here the quest would advance and the next Component and its "Full Condition State" would be selected. Quest Generator would then select the 1st path from the previous Component, with its GameState, to send to Quest Analyser along with the new Component and its "Full Condition State". This process repeated with the second path from the previous Component, and carried on until all paths were used. Then the generator repeated the process with the next Component, until no more Component remained.

After all Components were exhausted and all possible paths were discovered and recorded, the quest was then documented into a text file. This document included the quest's Components, Full Condition States, GameState at the end of each Components' queries, paths, objects' appearance rate, number of quests discarded during generation, and number of quests discarded during query. The

information was then used for analysing the quest and MPQ-Generator characteristic as a whole.

Component's Full Condition State

follow: [[player,sameLocation,merchant1,z,zz,zzz],[blacksmith1,isAlive,true,z,zz,zzz],[soldier1,isAlive,true,z,zz,zzz]]
follow: [[player,sameLocation,soldier1,z,zz,zzz],[blacksmith1,isAlive,true,z,zz,zzz],[soldier1,isAlive,true,z,zz,zzz]]
kill: [[soldier1,isAlive,false,z,zz,zzz],[blacksmith1,isAlive,true,z,zz,zzz]]
goto: [[player,sameLocation,lumberjack1,z,zz,zzz],[lumberjack1,isAlive,true,z,zz,zzz],[blacksmith1,isAlive,true,z,zz,zzz]]
report: [[player,sameLocation,blacksmith1,z,zz,zzz],[blacksmith1,isAlive,true,z,zz,zzz]]

Path ID: a1b2c1d1e1

start_new_component:no_action_need
start_new_component:no_action_need
start_new_component:

ac_hire_to_attack:	player, mob_npc_1, doctor1
char_die:	doctor1,
soldier_capture_criminal:	mob_npc_1, soldier1
criminal_escort_to_jail_also_lose_crime:	mob_npc_1, soldier1
ac_hire_to_attack:	player, thief1, soldier1
move_from_to:	jail, city, thief1
char_die:	soldier1
ac_move_from_to:	jail, city, player
move_from_to:	jail, city, player

start_new_component:

ac_move_from_to:	city, forest, player
move_from_to:	city, forest, player

start_new_component:

ac_move_from_to:	forest, city, player
move_from_to:	forest, city, player

Figure. 39: An example of a recorded [path] from a quest generated using "kill enemies" strategy.

CHULALONGKORN UNIVERSITY

5. Result and Analysis

5.1 Result

We analysed results from generated quests with the following strategies; [Kill enemies], [Steal Stuff], [Obtain luxuries]. A quest was set to have between 4 to 8 lengths of Component with maximum of 3 sequence of actions to complete each Component. Thirty quests were generated for each strategy. The highest possible number of paths is 136 paths from the strategy [Obtain Luxuries]. No range of acceptable number of paths was selected because we wanted to see the full range of possible number of paths for each strategy. No additional weight was given to any type of Components. All <Components> could be broken down into any of the possible set of <Components> and Components as stated within Table 4, with equal probability. No optimization was made to the generation procedure to enhance any specific property of the quest.

Analysing the [Kill enemies] strategy. This strategy could be completed in 30.3 different paths on average, with median at 28. See Figure. 40 for the distribution rate. On the other hand, strategies that focused on non-combat (killing) aspect showed lower number of possible paths on average. The strategy [Steal stuff] averaged only at 14.3 with the median of 8. This was likely caused by the limited possible interaction between items and characters. Items could only be gathered, looted from dead body, or traded for, while a dead character could result from multiple ways of chain-reaction between NPCs and players. However, if the items were in NPCs' possession, the number of paths was much higher than average, but not equal to the average number of paths from 'kill' Component. The number of paths was higher because NPCs could be killed in many ways to get the items. Table 10 shows that for quests in [Steal stuff] strategy, the action 'Pickup_body' appears more than 4 times compare to quests in [Obtain luxuries] strategy. For the action 'Pickup_body' to be valid, the target NPC must be killed. The amount of 'Direct_Attack' and 'Hire_to_attack' in strategy [Steal Stuff] are much higher compared to that in [Obtain luxuries], indicating attempts to kill NPCs. The reason the number of paths was lower than the number of paths from 'kill' Component was likely due to 'Pickup_body'

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

action exceeding the permitted number of actions per Component. Had the permitted number been higher, the number of paths would likely be equal or higher to that of a normal 'kill' Component. For [Obtain luxuries], such disparity was not shown. This was because the item was not usually in NPC's possession, hence the multiple ways to kill an NPC were not included in the paths.

Figure. 40 to Figure. 42 shows the disparity between all strategies and their generated quests' number of paths. The selected strategies include [Kill enemies], [Obtain luxuries], and [Steal stuff].

Recorded paths could also be listed and analysed by users. For example, our experiment generated an interesting [Kill enemies] path with a player bribing the city guard to arrest the city doctor and lock him in jail. Then the player successfully poisoned the target because no doctor was available. Another interesting path showed a player attacking a NPC without killing him to get captured and escorted to jail. The soldier also captured a thief (quest target who just escaped out of jail) while escorting the player to jail. Then the player killed the target and stole a lockpick from the target and used it to escape from jail and report the quest result.

These paths are interesting because they showed how MPQ-Generator is capable of discovering complex actions planning. In the path where a player attacked an NPC to get captured, the 'attack' did not directly result in the death of Thief, or player being able to 'attack' Thief. The changed in conditions that result in player being able to directly attack Thief came from a 'chain reaction' of the dynamic environment and how NPCs react to each other.

Table. 10: This table shows how many numbers of each action appeared in paths for each strategy.

	Strategy		
Actions	**Kill Enemies**	**Obtain luxuries**	**Steal Stuff**
Direct_Attack	586	167	587
Hire_to_Attack	2158	442	961
Pickup_body	69	111	457

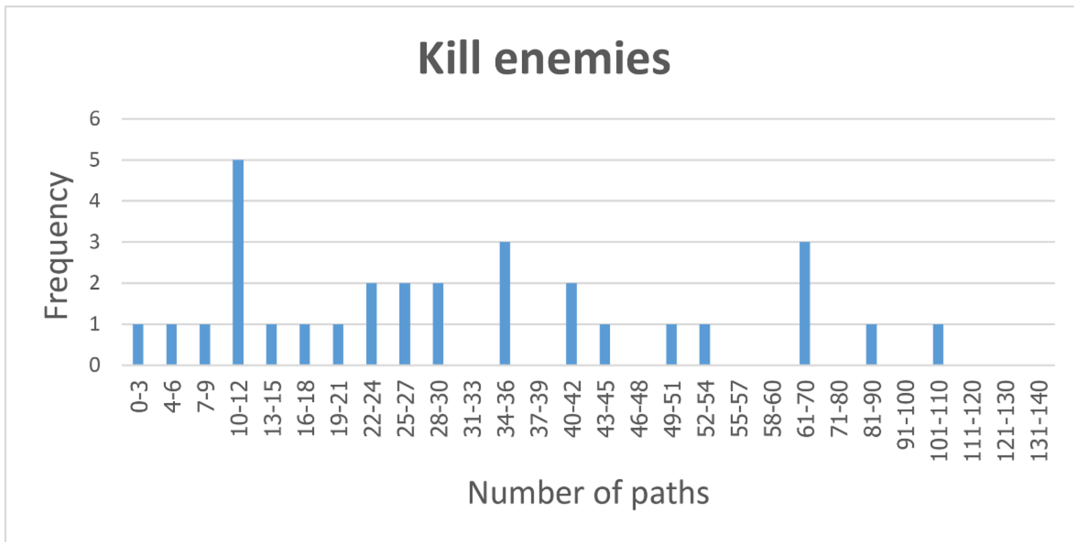


Figure. 40: Quests statistic for Kill enemies strategy, from 30 generated quests.

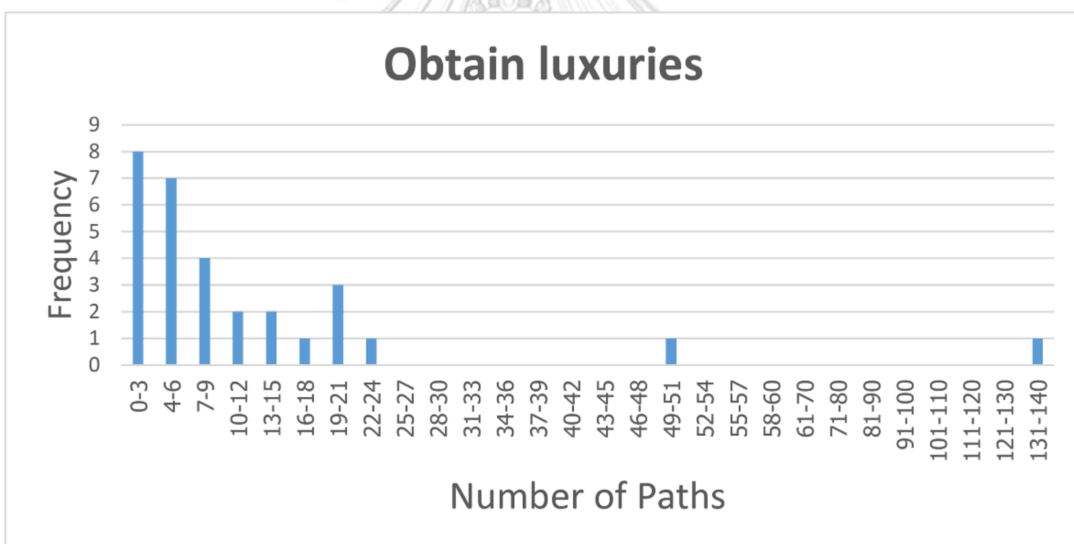


Figure. 41: Quests statistic for Obtain luxuries strategy, from 30 generated quests.

Figure. 42: Quests statistic for Steal stuff strategy, from 30 generated quests.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

5.2 Comparison

This thesis used a modified ReGEN's metric system as an evaluation tool. Quest generated using the proposed algorithm were compared with quests from other sources. Figure. 43 shows how ReGEN metrics are used to evaluate quests from Skyrim, Radiant system and Witcher series. The comparison in Figure. 43 uses average values from all quests in each game. In this work, we calculated our quest characteristics using ReGEN's formulas, making our quest comparable to ReGEN's analysis. The comparison was made to determine how our quests' characteristics were different from other systems. It was not a statement that our work was better than others in a specific field. Some metrics were marked "Not Applicable" because the work in this thesis was not designed for such metrics and had no available information to calculate the metrics.

Table 11 shows a modified ReGEN's metric table used to compare MPQ-Generator's quests characteristics against quests from other systems. The MPQ-Generator's values came from the analysis of 90 generated quests from the previous section. No modifications or specific designs were made to the generation system to optimise performance for any metrics.

It must be noted that the system with the highest value was not the 'best' system in every situation. It depended on what were required from the quest. For example, a designer may want a system with high 'Cost' to create quest with higher impact to game world upon completion, while another designer may want a system that has zero 'Cost' so that quests could be generated with consistent quality.

Metric	ReGEN	SQUEGE	Radiant Quests	Skyrim Main Quests	Witcher Main Quests
Narrative Content	5.32 ± 0.90	5.08 ± 3.25	2.33 ± 0.85	5.12 ± 2.37	15.41 ± 9.52
Longest Path	4.75 ± 0.54	4.13 ± 1.64	2.17 ± 0.80	5.12 ± 2.37	12.09 ± 6.41
Shortest Path	4.43 ± 0.59	3.04 ± 0.79	2.17 ± 0.80	5.12 ± 2.37	11.18 ± 6.58
Average Path	4.59 ± 0.52	3.78 ± 1.29	2.17 ± 0.80	5.12 ± 2.37	11.61 ± 6.46
Most Branches	0.57 ± 0.50	1.63 ± 1.07	0.17 ± 0.37	0	1.26 ± 1.09
Fewest Branches	0.57 ± 0.50	1.17 ± 0.37	0.17 ± 0.37	0	1.26 ± 1.09
Average Branches	0.57 ± 0.50	1.48 ± 0.88	0.17 ± 0.37	0	1.26 ± 1.09
Highest Cost	0.84 ± 0.55	1.42 ± 0.91	0	0.29 ± 0.46	0.21 ± 0.40
Lowest Cost	0.43 ± 0.50	1.29 ± 0.93	0	0.29 ± 0.46	0.12 ± 0.32
Average Cost	0.63 ± 0.42	1.38 ± 0.89	0	0.29 ± 0.46	0.17 ± 0.34
Most Encounters	0.25 ± 0.43	0	0.42 ± 0.57	0	1.44 ± 1.90
Fewest Encounters	0.0 ± 0.0	0	0.42 ± 0.57	0	1.24 ± 1.88
Average Encounters	0.12 ± 0.22	0	0.42 ± 0.57	0	1.33 ± 1.89
Highest Uniqueness	0.95 ± 0.09	0.68 ± 0.14	0.95 ± 0.12	0.72 ± 0.20	0.59 ± 0.25
Lowest Uniqueness	0.94 ± 0.09	0.58 ± 0.10	0.95 ± 0.12	0.72 ± 0.20	0.53 ± 0.24
Average Uniqueness	0.94 ± 0.09	0.62 ± 0.09	0.95 ± 0.12	0.72 ± 0.20	0.56 ± 0.24
Narrative Richness	0.03 ± 0.12	0	0	0	0.03 ± 0.16

[Figure. 43] ReGEN's metrics applied to quests from multiple games and systems [47].

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Table. 11: This table shows how quests from MPQ-Generator perform compared to quests from other systems. The results from ReGEN to Witcher Main Quests are taken from Figure. 43

Metric	MPQ-Generator (2019)	ReGEN (2014)	SQUEGE (2007)	Radiant Quests (2011)	Skyrim Main Quests (2011)	Witcher Main Quests (2007)
Narrative Content	4.85	5.32	5.08	2.33	5.12	15.41
Average Path	4.85	4.59	3.78	2.17	5.12	11.61
Average Branches	0	0.57	1.48	0.17	0	1.26
Average Cost	N/A	0.63	1.38	0	0.29	0.17
Average Encounters	N/A	0.12	0	0.42	0	1.33
Average Uniqueness	0.90	0.94	0.62	0.95	0.72	0.56
Narrative Richness	N/A	0.03	0	0	0	0.03
Weight of Choice	0.68	N/A	N/A	N/A	N/A	N/A

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

5.2.1 Measurement and Evaluation Calculation

Narrative Content

One unit of Narrative Content was equal to 1 Component. This represented the number of events that happened while doing a quest. In a quest with branching path, Narrative Content counted all nodes regardless of whether a player was able to experience all of them in 1 playthrough. The Narrative Content of MPQ-Generator's quests were determined by the quest size configuration. The sample shown in this thesis used 4-8 Components size quest setting and the actual average size was 4.85 units (from 4 types of quests in the experiment). This was close to ReGEN, SQUEGE, and Skyrim Main Quest's Narrative Content. The Witcher had a much larger Narrative Content value. To improve MPQ-Generator's Narrative Content, users can configure the quest's size to be bigger than 4-8 Components used as sample sets in this thesis.

Path

Path was defined as Narrative Content of each branch of the story. If a quest did not have branching, its Path would be equal to its Narrative Content. There was no branching in the quest generated from MPQ-generator, therefore players would experience all of the Narrative Content according to this definition.

Not to be confused with Path defined in this thesis as 'set of action players can take to complete a quest.'

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

Branch

Branch was defined as the number of times a player was able to choose quests' objectives. Therefore, it was not the same as achieving the quest's objective via different quest paths. In this definition, MPQ-Generator's Branch metric was zero because there was no quest where players needed to choose the quests' objectives.

Cost

When a non-renewable object was permanently removed from game world, it counted as 1 Cost (removing objects generated along with the generated quest did

not count). Cost was defined as the consequence of finishing a quest that reduced the flexibility of generating a new quest. In MPQ-Generator, objects might change properties such as turning from being ‘alive’ to ‘dead’ and vice versa. However, no action would result in permanent deletion of an object. Therefore, this metric was not applicable.

Encounter

Similar to Cost but used for renewable objects such as items or monsters that spawned periodically at certain locations, or NPC generated for the quest. MPQ-Generator did not simulate spawned monster/item/NPC encounter or combat. Therefore, MPQ-Generator could not be compared with other systems using this metric.

Uniqueness

Uniqueness was defined as a measurement of the variations of the quest tasks. It was calculated by dividing the number of unique Components with the total number of Components (Narrative Content). For example, if a quest contained 2 [Goto ‘x’] Components and 1 [Kill ‘z’] Component, they were counted as 2 units of unique Components and 3 units of Narrative Contents. The Uniqueness value of such quest would then be 2/3 (0.66). MPQ-Generator quest’s average Uniqueness (from 3 types of quests in the experiment) was 0.90 which was higher than SQUEGE’s (0.62), Skyrim’s Main Quest’s (0.72), or even The Witcher’s Main Quests (0.56). Quests generated from MPQ-Generator contained more diverse activities and goals than these quests.

Narrative Richness

This metric measured the amount of unintentional consequences of a narrative/quest that led to another narrative/quest. By unintentional, it meant ‘not as part of Component/quest goal state or reward.’ The more quests generatable from that consequences, the higher the Narrative Richness became. However, MPQ-Generator did not generate consecutive quests from the final state of any quest.

Therefore, the Narrative Richness of quests generated from MPQ-Generator was not measurable.

Weight of Choice

Weight of Choice measured the effect of player's choices by comparing how different all final states of the game were at the end of the quest. The lower the number, the more different the final states were from others. The measurement was calculated from the average differences of each generated quest's GameState at the end of each paths. The difference (intersect) of each path was divided by the union of each path. There seemed to be a correlation between weight of choice and the number of paths. This metric showed the gravity of player's choice of actions and how the end state of the game could be different depending on the chosen sequence of actions. The lower the Weight of Choice, the more different each final GameStates were from each other. Each quest strategies had the following Weight of Choice: [Obtain luxuries = 0.74], [Steal item = 0.72], and [Kill enemies = 0.60].

It could be seen that quest's strategy with higher number of paths had lower Weight of Choice. The other systems' Weight of Choice were not available because, as mentioned in [47], complete game world information was not available. Therefore, it was not possible to compare MPQ-Generator's Weight of Choice with other systems.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

5.3 Path analysis

We turned generated quests' paths into trees for analysis. Each node represented actions performed by a player in each path, excluding the 'start_new_component' node which indicated a start of new Component. At the start of each tree, paths would be represented by 'start_new_component' equal to the number of paths in that quest. If two or more paths pointed to the same nodes, it meant that at that level of action and Component, both parts performed that same action. There was also special node that represented "player performing either action A, or action B in any order" that was created every time two or more paths shared a group of action. Such as:

Path 1: [ac_direct_attack, Thief1] + [ac_direct_attack, Merchant1]

Path 2: [ac_direct_attack, Merchant1] + [ac_direct_attack, Thief1]

Path 3: [ac_direct_attack, Merchant1] + [ac_direct_attack, Thief1]

Here the special node that the three paths pointed to would be labelled as shown below, where the number after 'count' indicated how many of that combination of action happened.

[direct_attack/Thief1-direct_attack/Merchant1 count = 1 ____ direct_attack/Merchant1-direct_attack/thief1 count = 2]

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

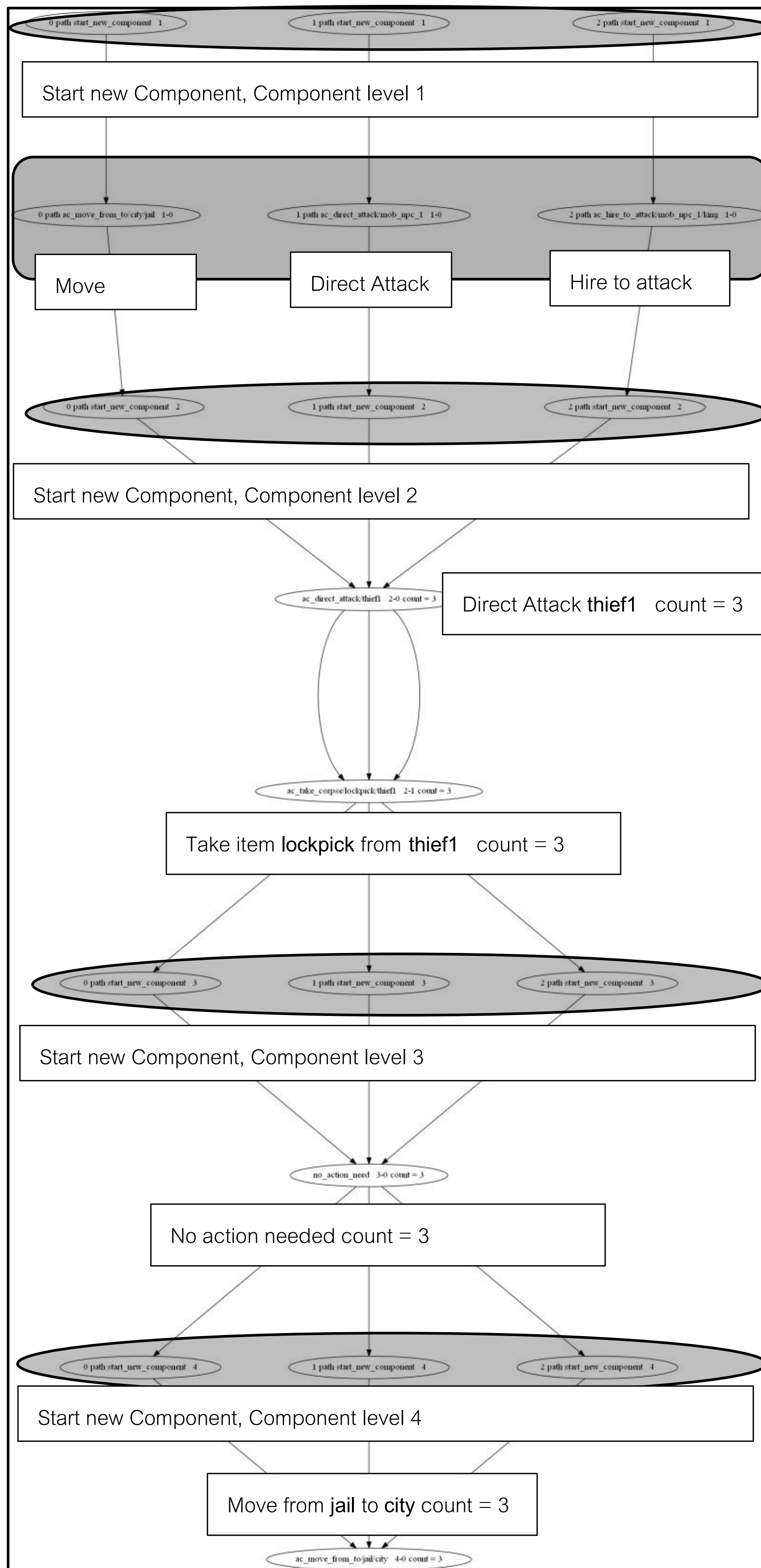


Figure. 44: A tree representing paths of a quest from 'Obtain luxuries' template.

Figure. 44 shows a simple quest tree generated from ‘Obtain luxuries’ template. Nodes with the same action and subject of action were grouped as a single node. The quest’s Components consisted of [follow: thief1], [take: lockpick], [wait], and [report: doctor1]. This quest could be completed with 3 different paths.

The nodes inside the topmost highlighted oval represented the starting of each path. They all were “start_new_component_1” nodes. Then in the highlighted rectangle just below, each “start_new_component_1” node pointed to different nodes. These nodes in the highlighted rectangle represented 3 different actions that could lead the player to quest completion. They were [Move_to / jail], [Direct_attack / mob_NPC_1], and [Hire_to_attack /mob_NPC_1 /king]. All of these actions and subsequent reaction got the player to jail, the same place as thief1.

The next part of the tree pointed to nodes “start_new_component_2”. These nodes are highlighted in oval in the figure. These component nodes told us that the previous component’s objectives were successful, and the next component could be started.

The quest had 4 level of Components. The 4 sets of “start_new_component” are shown within oval shape highlights.

All “start_new_component_2” nodes pointed to the same node. This showed that these paths shared the same action which led to quest completion. All 3 paths shared the action “Direct_attack / Thief1” node. This meant that if the quest was to progress ahead, the only option was to perform ‘Direct_attack’ on the NPC Thief1. After the “Direct_attack / Thief1” node, all subsequent nodes and their arrows converged into a single node, expanded into 3 “start_new_component” then converged again.

Thus, for this quest, all the ‘variety’ in how to complete this quest only stemmed from the first Component. Only in the first Component that player had ‘3’ choices to choose on how to complete the Component’s objective. In the subsequent Components, there was only 1 node that led to the next node and no divergence existed.

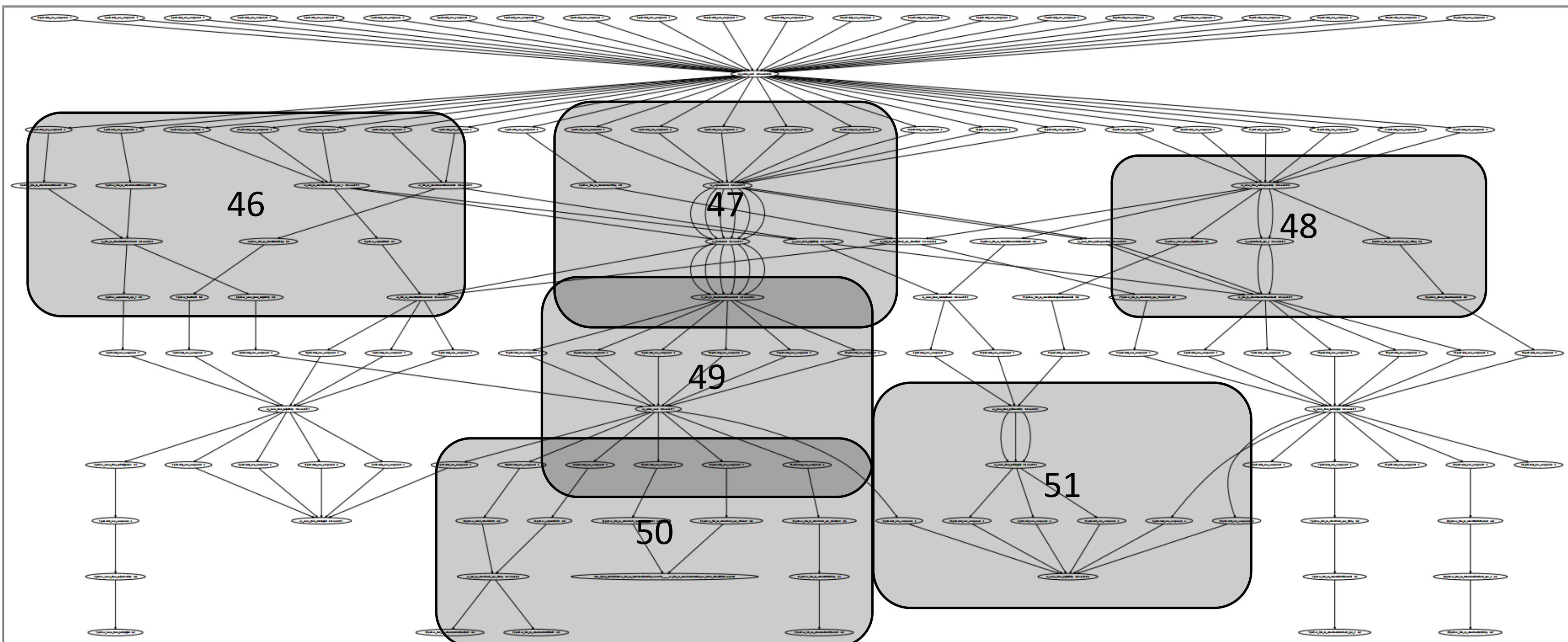


Figure. 45: A tree representing paths of a quest from ‘Kill enemies’ template, the highlighted areas are shown in other figures.

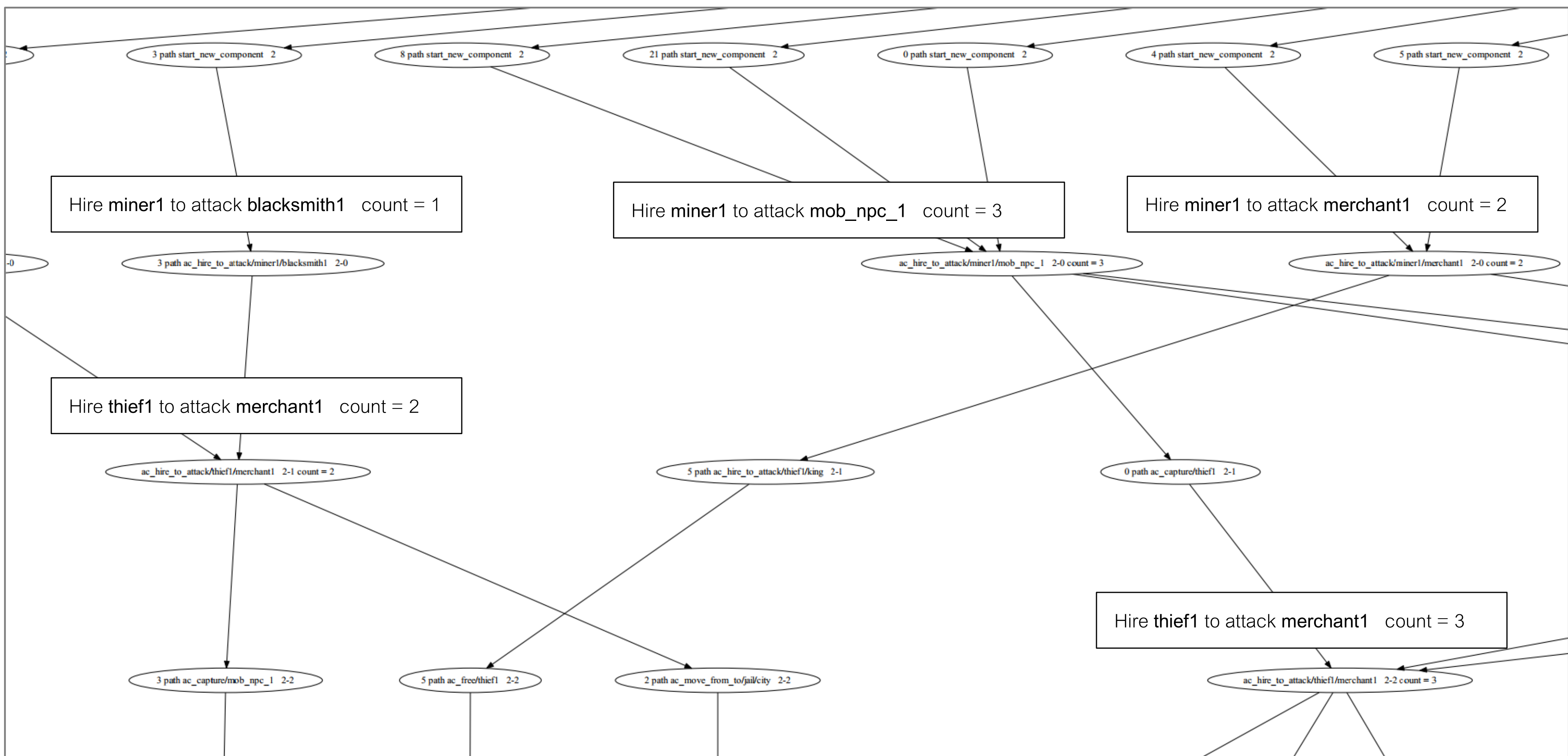


Figure. 46: The left part of the tree from Figure 44.

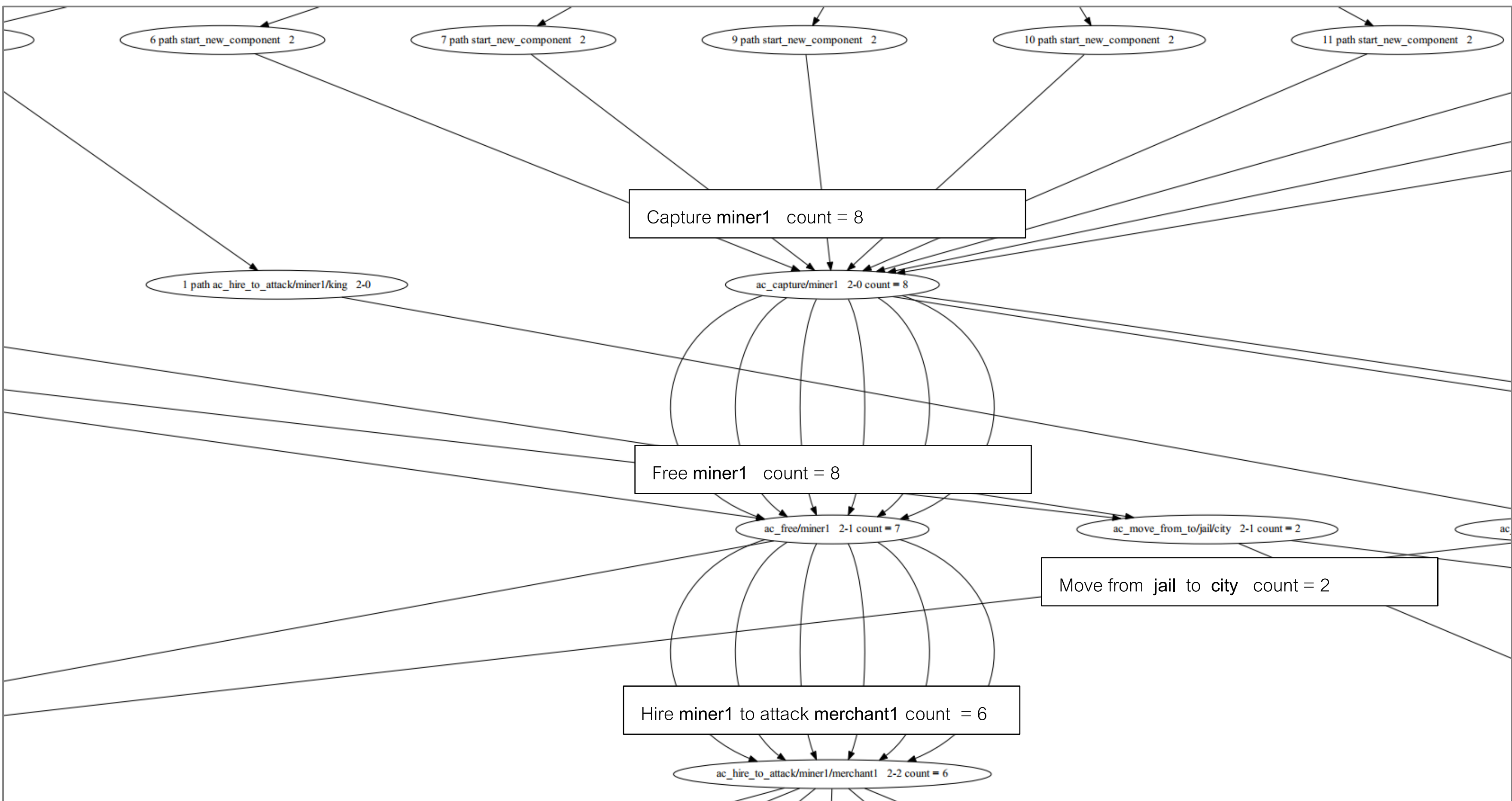


Figure. 47: The middle part of the tree from Figure 44.

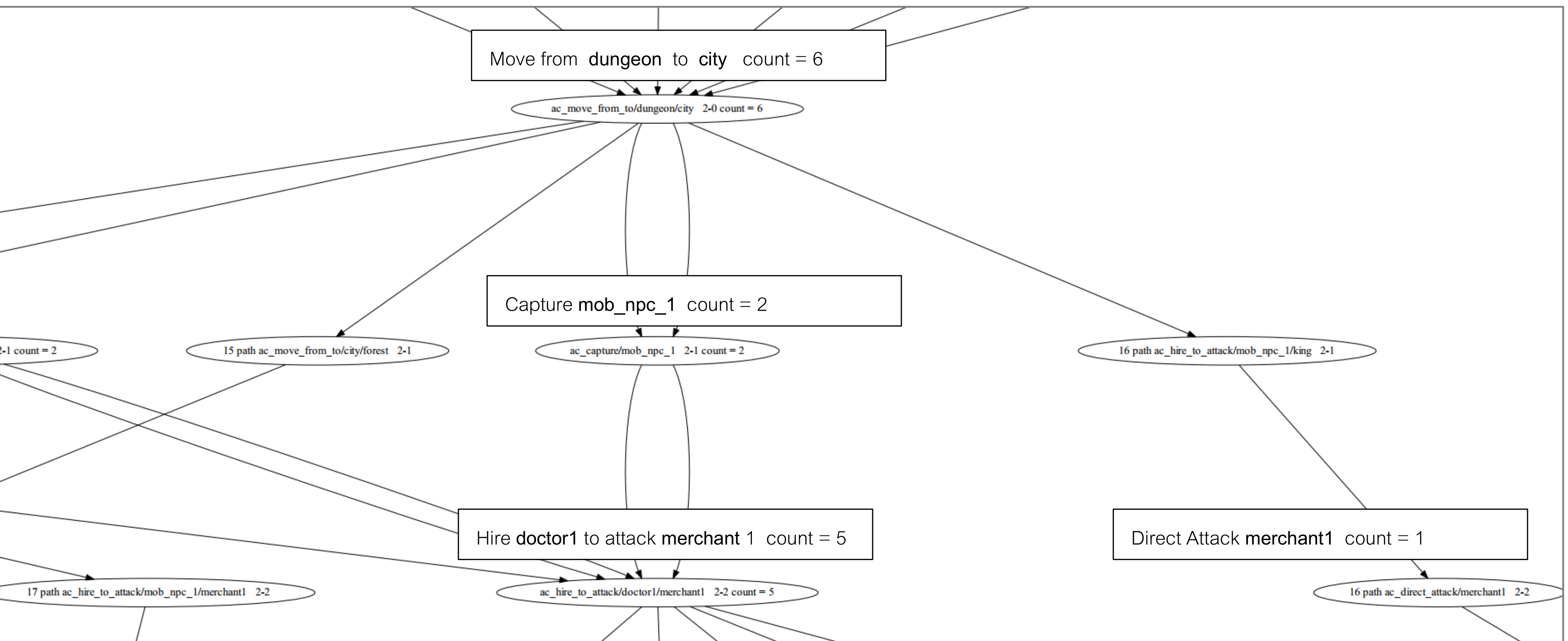


Figure. 48: The right part of the tree from Figure 44.

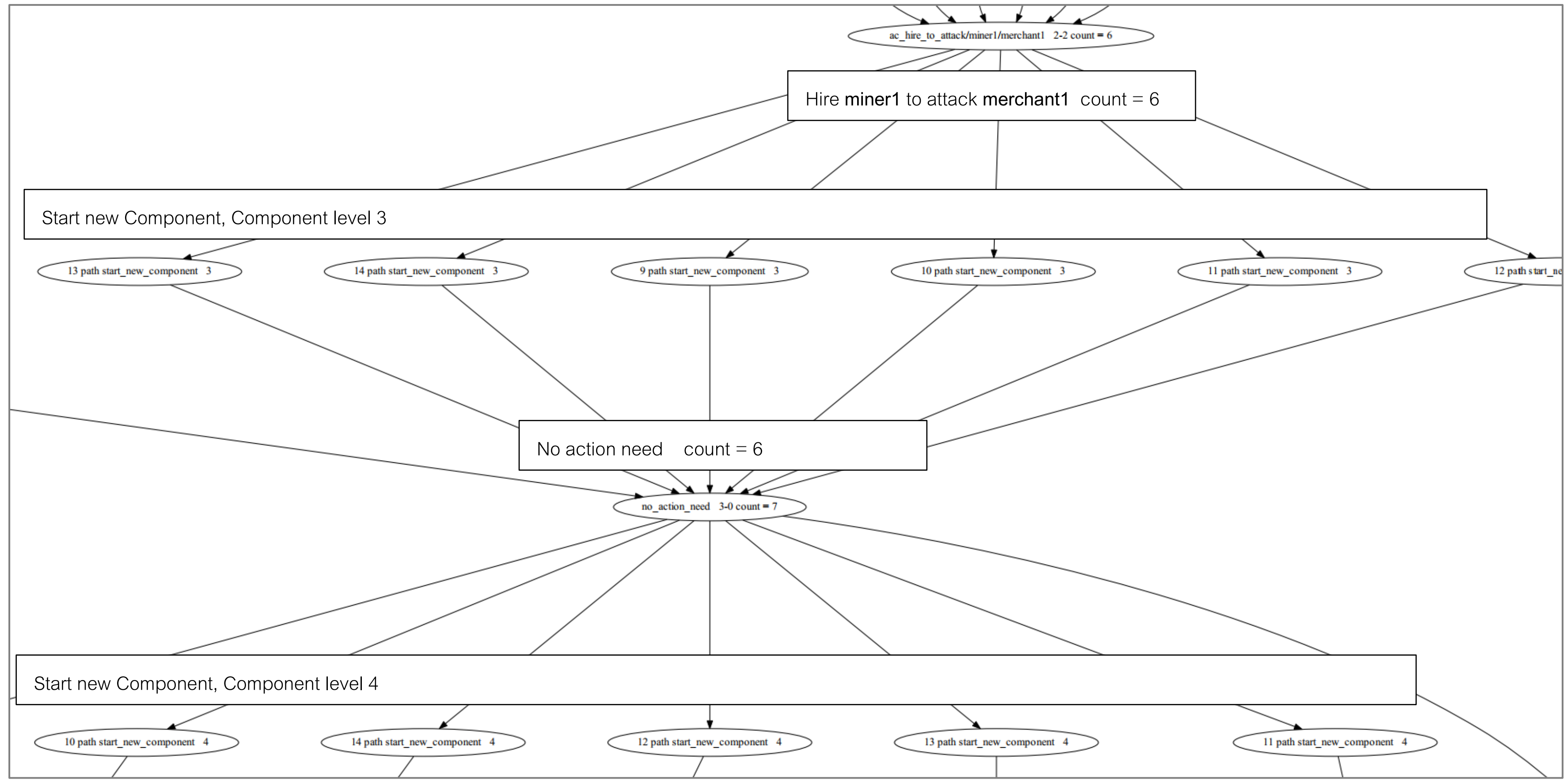


Figure. 49: The middle-bottom under the capture miner part of the tree from Figure 44.

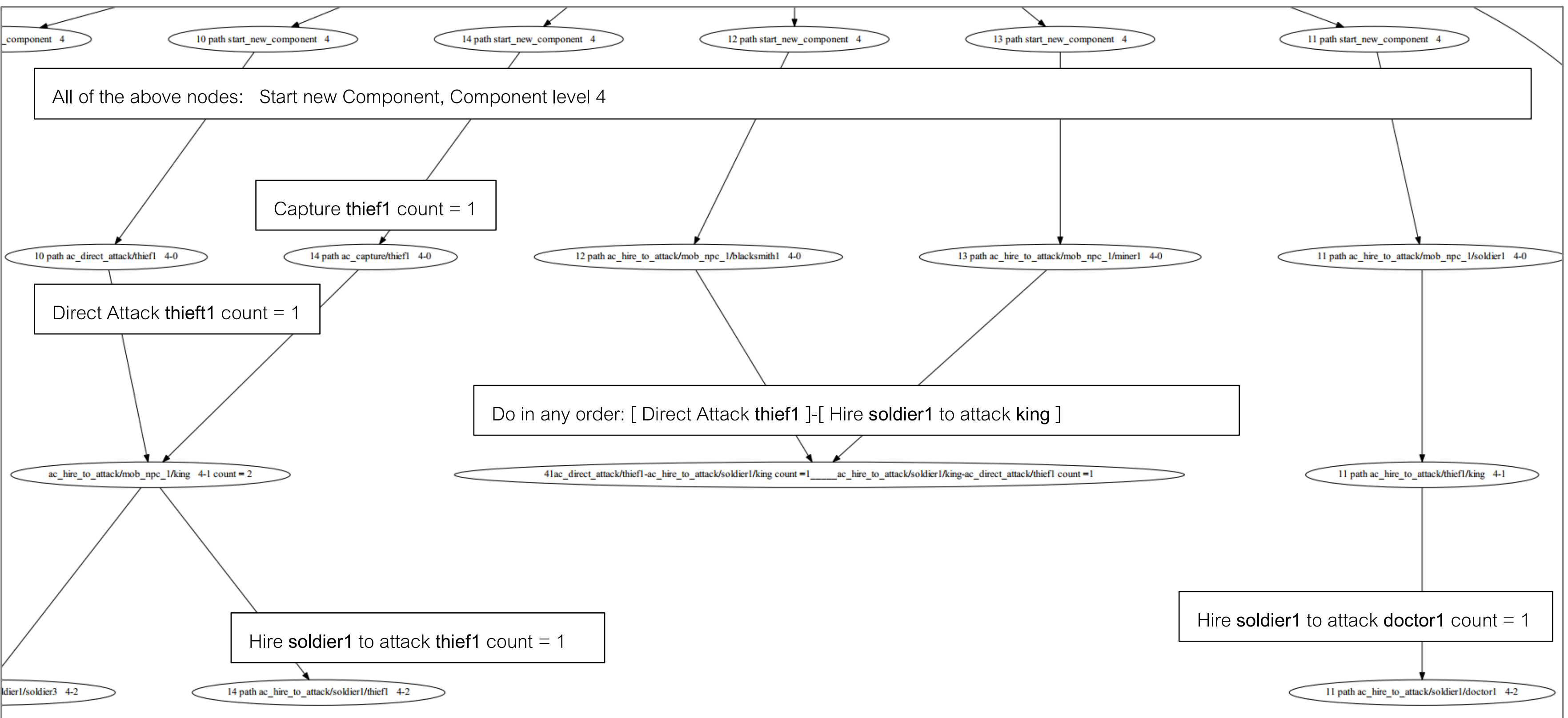


Figure. 50: The bottom part of the tree from Figure 44 showing that some paths contain seemingly random, non-related actions.

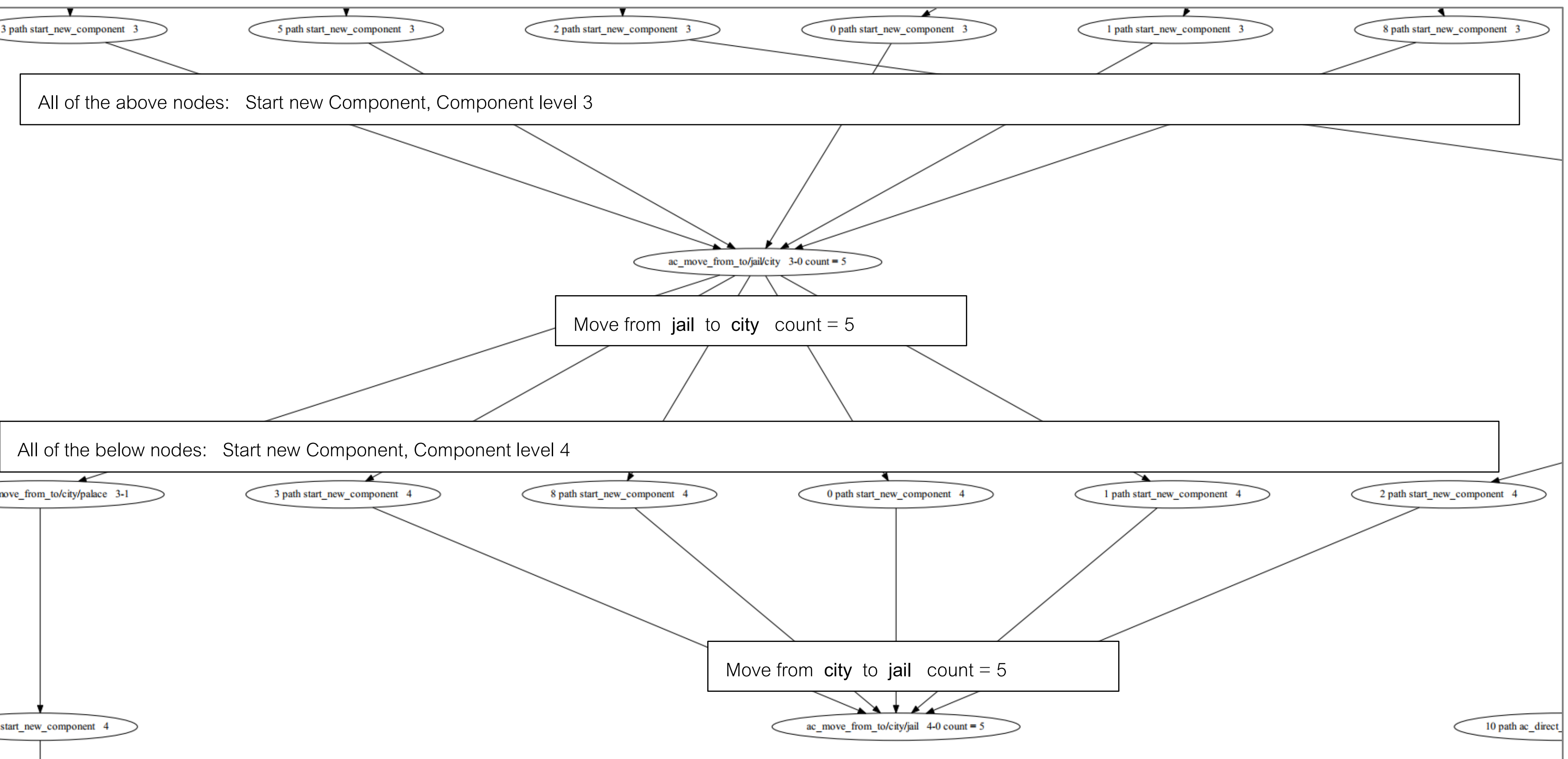


Figure. 51: The bottom right part of the tree from Figure 44.

Figure 45 shows a tree of a quest generated using “Kill enemies” strategy. The quest had 19 paths of completion and consisted of the following Components: [stealth: PLAYER], [kill: Merchant1], [explore: Theif1], and [report: Soldier1].

From Figure 45, at the 1st Component level, all nodes (paths) pointed to ‘no_action_need’ node which indicated that the Components’ conditions were already satisfied when the query system reached those Components. This was because stealth was not simulated and the condition was considered satisfied as soon as the player was at the same place as the target, which the player was. Then in the 2nd Component, in the middle of the tree, 6 of the paths pointed to ‘capture miner1’ then ‘free miner1’ then ‘hire_to_attack/miner1/merchant1’ (player hired Miner1 to kill Merchant1), as shown in Figure 47. Figure 46 and Figure 48 show that most of those paths contain the ‘hire_to_attack’ with the target being Merchant1.

While Figure 46 and Figure 47 used the same action to complete the 2nd level Component (hire_to_attack) and looked like these actions would result in the same GameState. In fact, they differed. The ‘capture-miner’ (Figure 47) had its 3rd level Component with ‘no_action_need,’ as shown in Figure 49, while the Component in Figure 46 had its 3rd level Component followed by either ‘goto Jail’ or ‘goto City’ action. This shows how performing action not related to the dead of Merchant1 setup different GameStates where the conditions allowed player to perform different actions to complete the next Component’s objectives.

The final Component’s (4th level Component) objective was to report to ‘Soldier1’ who was normally stationed at ‘City’, but due to ‘illegal’ activity he should now be stationed at Jail after Soldier1 escorted criminal there. The quest should have ended with “no_action_need” for the paths that started with capturing Miner1. However, some unnecessary actions were performed to complete the quest for those paths. For example, going to City then going back to Jail, or hiring Soldier1 to kill the king, which was followed by Soldier1 being captured and sent back to Jail. Some examples were shown in Figure 50.

We analysed and concluded that this quest should only consist of 7 to 10 paths at most, with the main difference stemming from either capturing miner to get to jail beforehand or skipping the capturing and hire someone to kill merchant1. Half

of the 19 paths were 'noised' path that were not removed during the path filtering using the previously mentioned 5 criteria. Since performing unnecessary actions at the 4th level Components changed the GameState and Path's action structures, the changed were major enough for the noised paths to get pass the filtering and be recorded.

However, the 5 criteria already reduced the number of paths from 45000 paths to around 6-20 paths (at one Component). Subsequent attempts to reduce noised paths resulted in multiple of the genuinely unique paths being removed and reduced the path number to only around 1-3 paths. Any future attempt at configuring the path criteria should be done with multiple set of criteria that would be used exclusively depending on the type of quests being queried.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

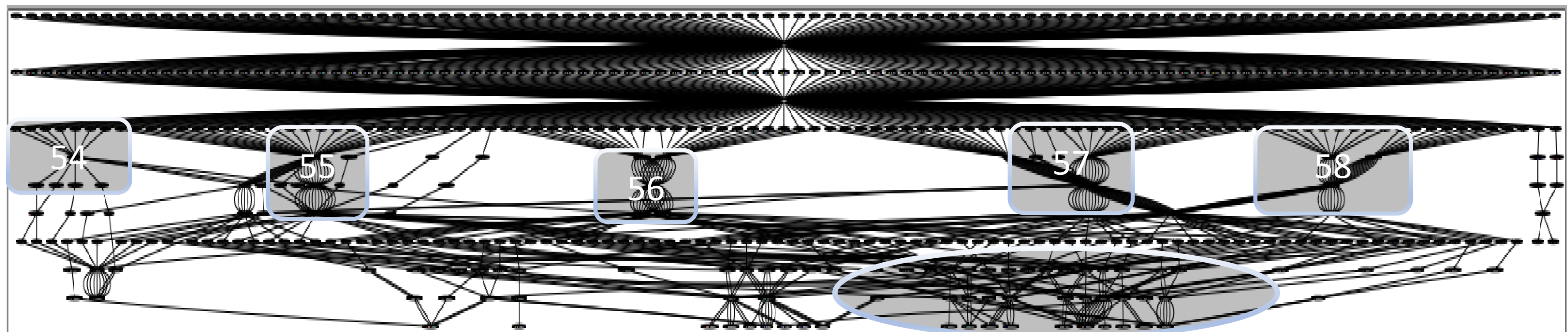


Figure. 52: A tree representing paths of a quest from 'Kill enemies' template. This tree was constructed with filter that used the object of each action to identify each node. The highlighted areas are shown in other figures.

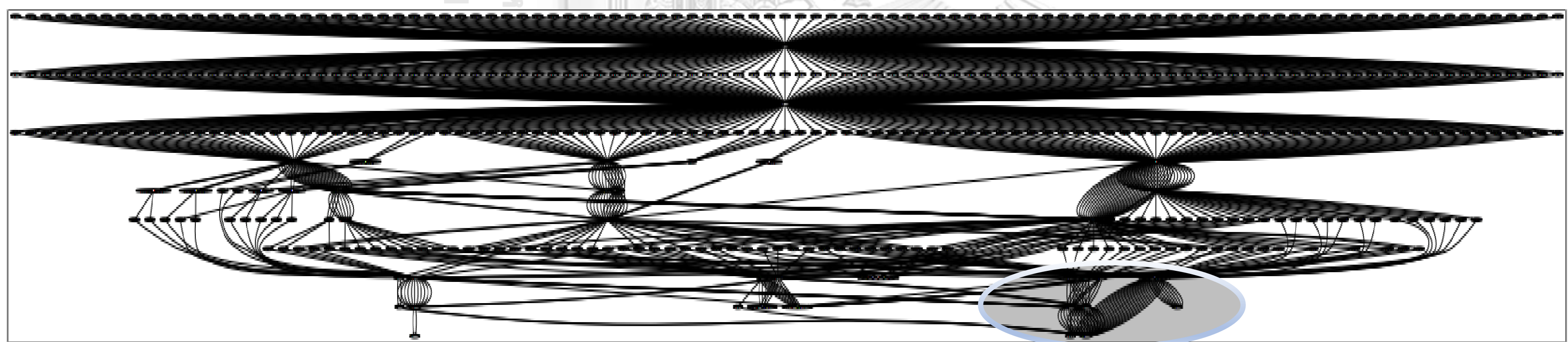
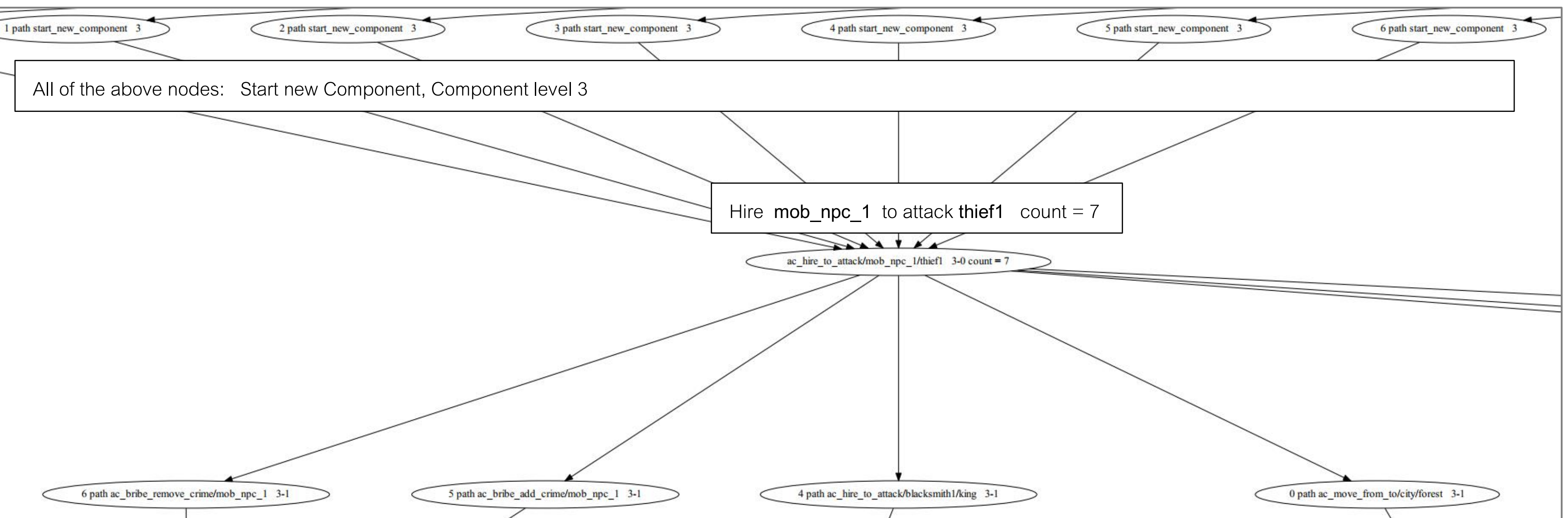


Figure. 53: A tree representing paths of a quest from 'Kill enemies' template. This tree was constructed without using the subjects of each action.



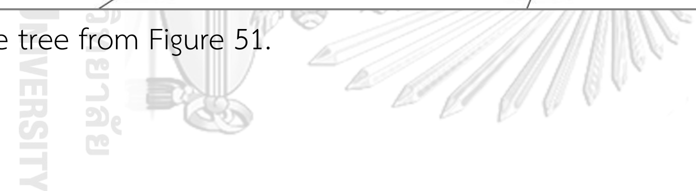


Figure. 54: The left-most group of the tree from Figure 51.

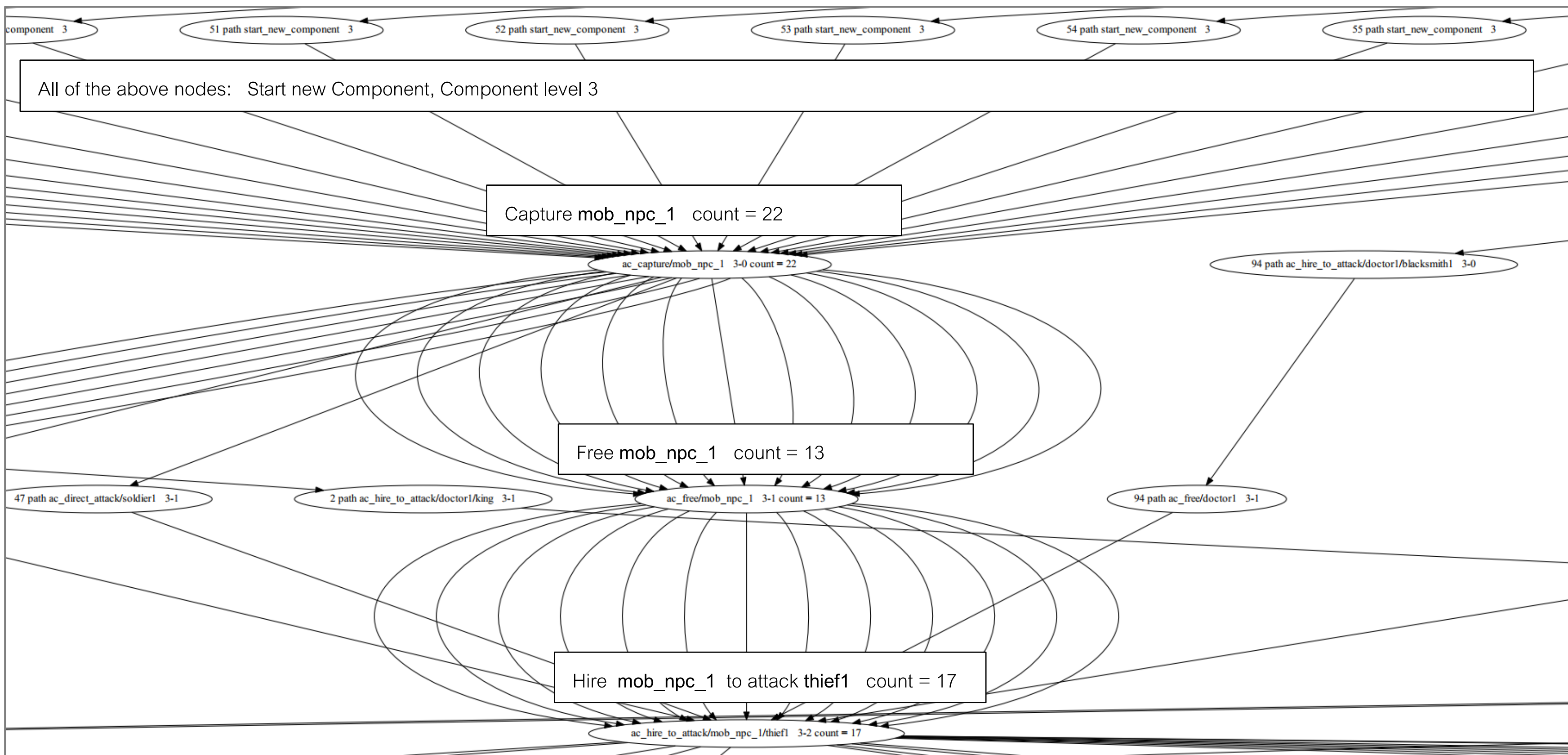


Figure. 55: The middle-left group of the tree from Figure 51.

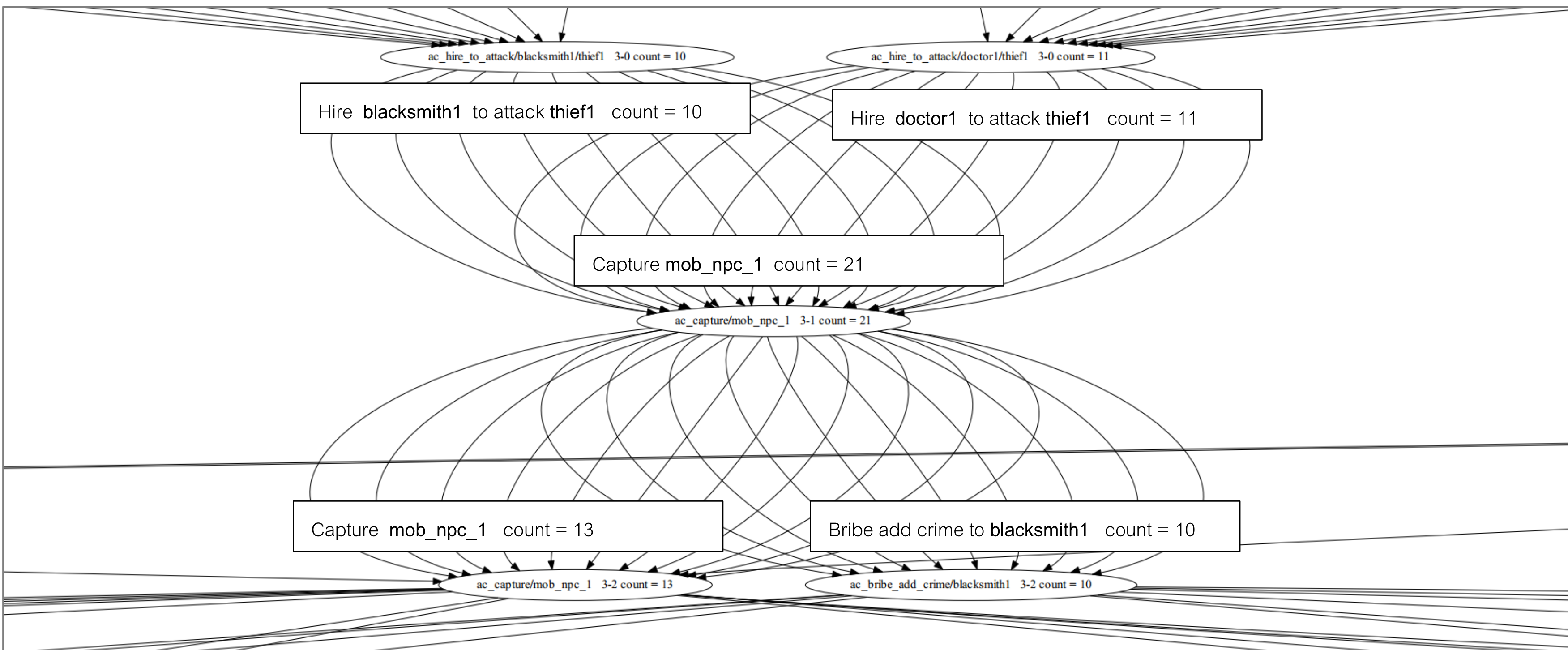


Figure. 56: The middle group of the tree from Figure 51.

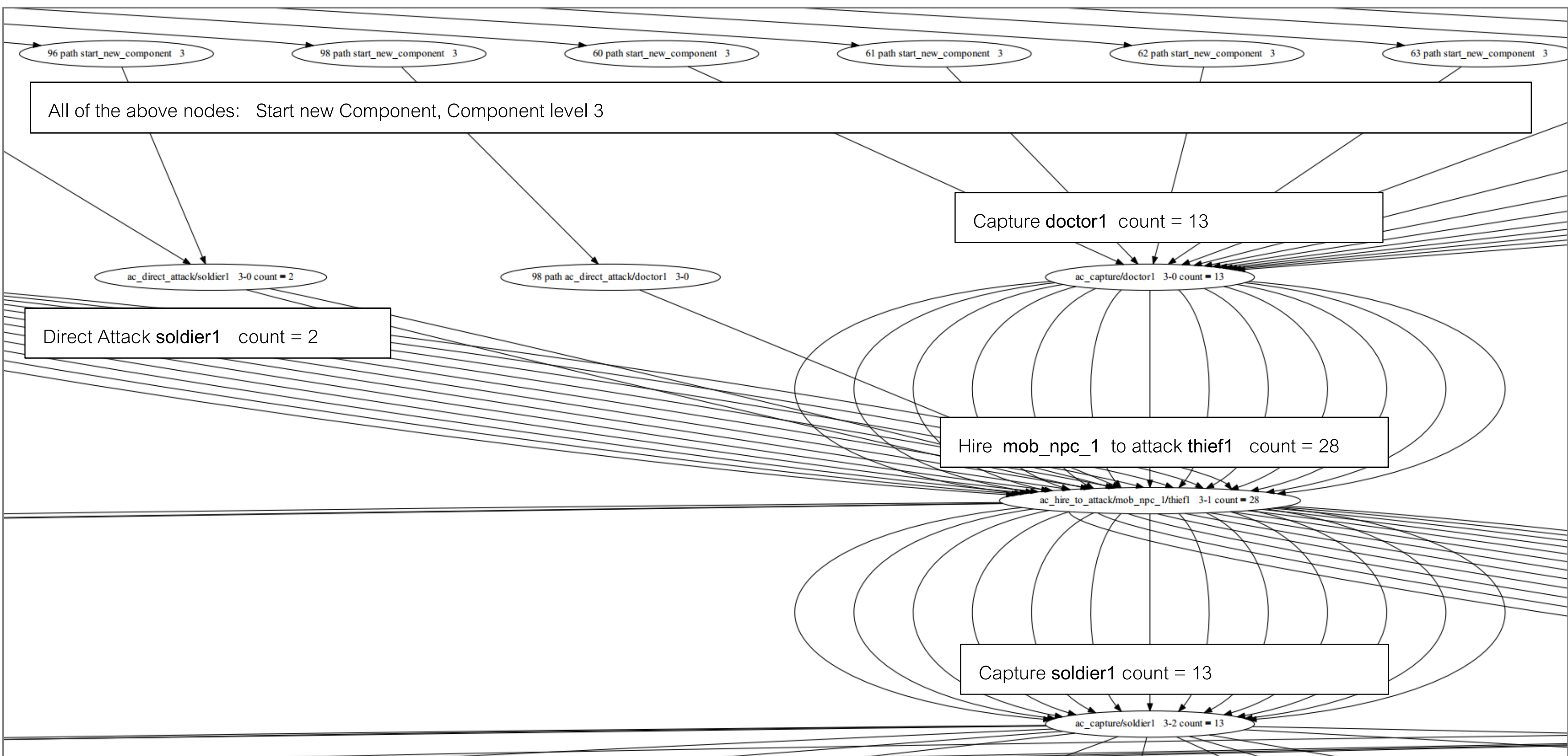


Figure. 57: The middle-right group of the tree from Figure 51.

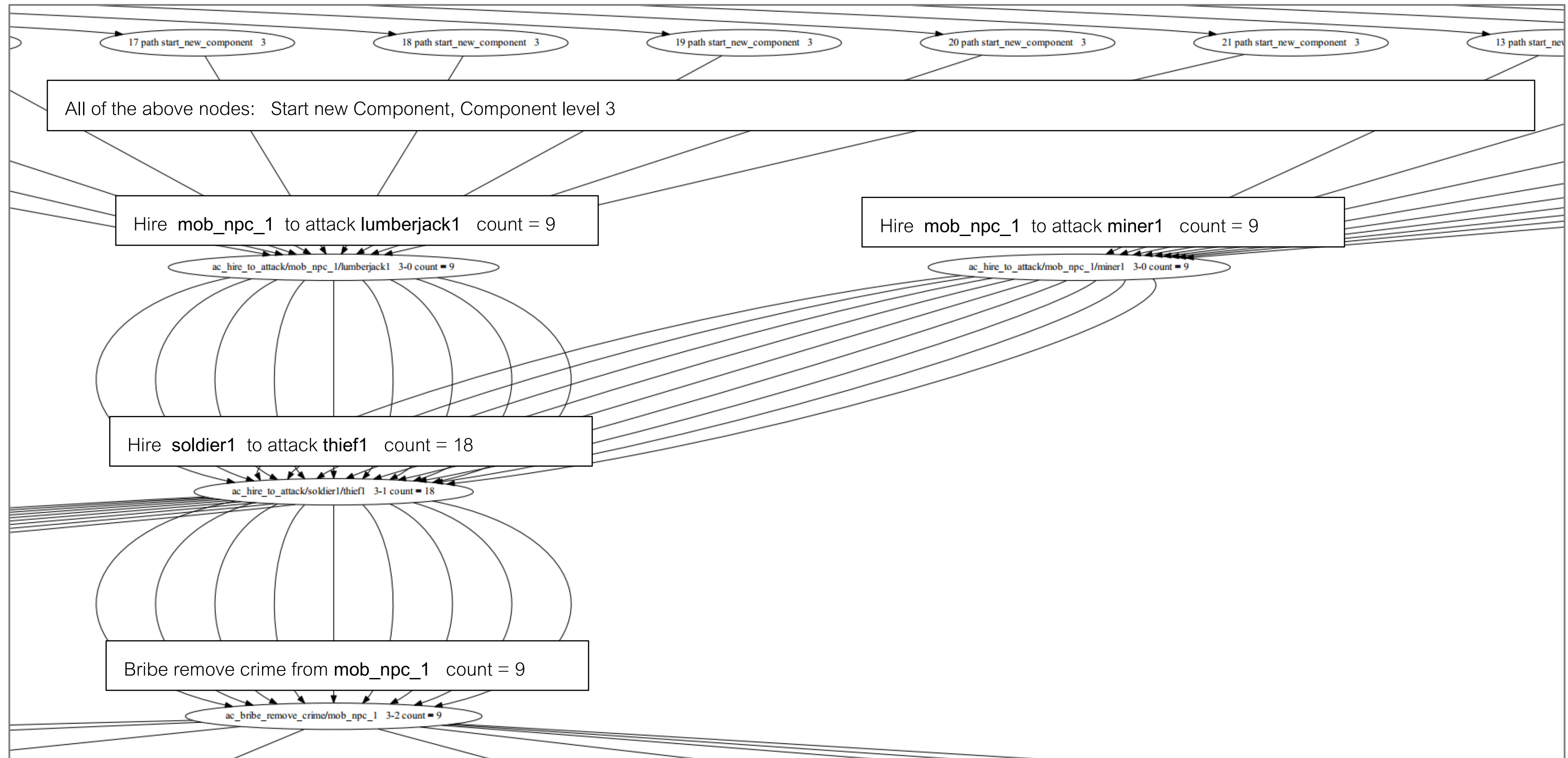


Figure. 58: The right-most group of the tree from Figure 51.

Another example can be seen in Figure 52 and Figure 53. It was also a quest from ‘Kill enemies’ strategy. The tree in the figures had 101 paths of completion with only 4 Components, [wait], [stealth: Mob_npc_1], [kill: Theif1], and [report: Soldier3]. The first two Components only pointed to a single node, “no_action_need.” This meant that the path variety effectively came from finding a way to complete only the remaining 2 Components.

Looking closer at Figure 52, in the 3rd level Component, most of the paths could be grouped into 6 groups that pointed to the same nodes as other paths in their own group, except for the right-most group. The paths in the right-most group (right beyond the box number 57 in Figure. 52) were different from paths in other groups because of their actions in the 4th level Components, or the lack thereof. The paths of the right-most group resulted in a GameState that completed the 3rd and 4th level Components’ objectives simultaneously. As shown in Figure 54, Figure 55, Figure 56, Figure 57, and Figure 58, it was almost certain that most of the actions that all the groups shared was “hire_to_attack” with the target being Thief1, the target of the [kill] Component. However, these look-alike paths were not grouped together into a single node because the NPC that were hired were different.

Would it be better if we considered all the ‘hire_to_attack’ nodes as duplicates? A tree could then be further simplified for analysis.

Let us compare this with the path merging that only took player’s actions into consideration. Figure 53 shows the tree from the same quest, but this time subject of each action was ignored. It can be seen that there was only 3 distinctive path groups, the one that started with ‘hire_to_attack’ node, the one that started with ‘capture’ node, and the ‘capture or hire_to_attack in any order’ node, with a little ‘direct_attack’ and other nodes between them.

Analysing Figure 53, it is possible to conclude that this Component has only 3 truly distinctive actions to advance the quest to the next Component. However, Figure 52 shows that once we took the subjects of actions into consideration, there were 5-6 unique combination of actions that could lead to the next Component. To determine which were better, we must look at how each of these set of actions affected player’s actions in Component level 4.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

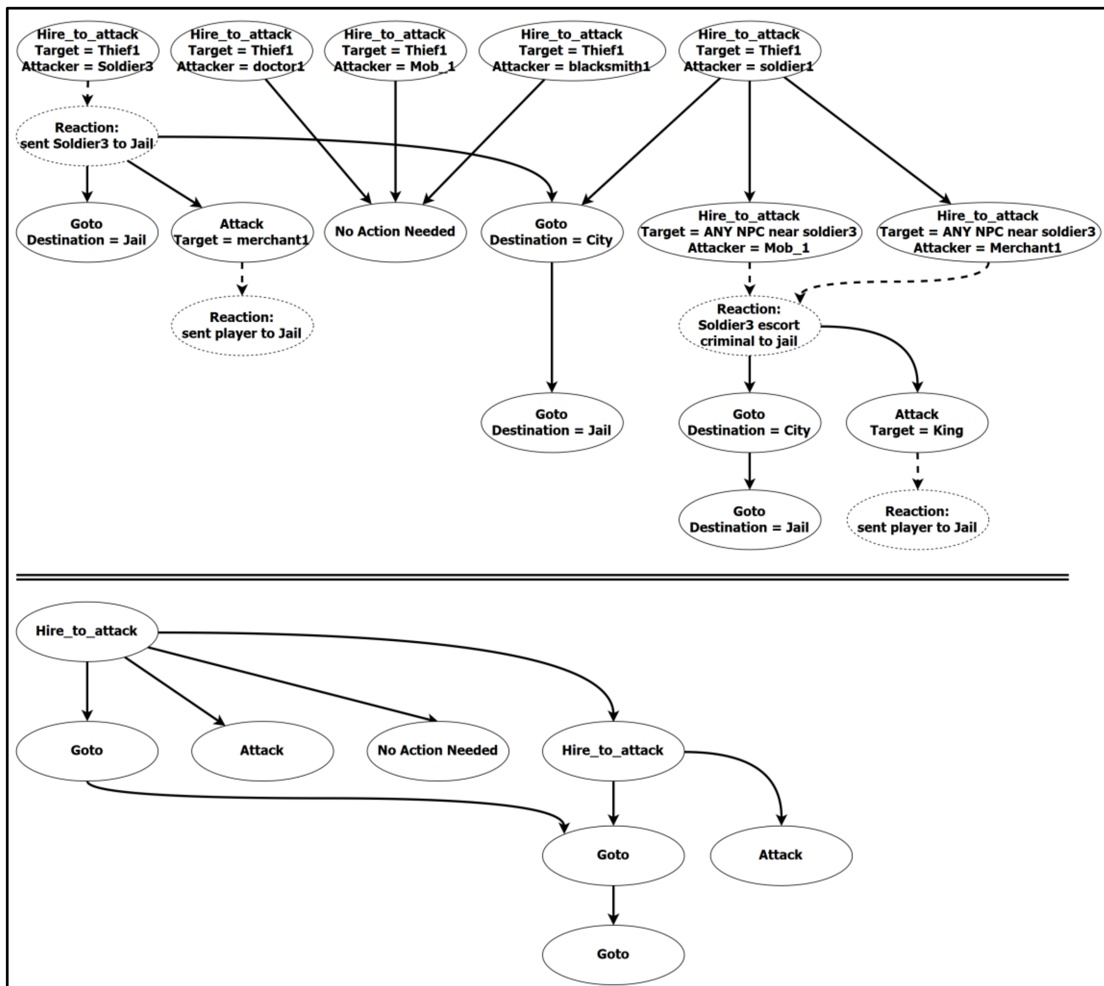


Figure. 59: A simplified version of the oval highlighted area from both Figure 52 (top) and Figure 53 (bottom).

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

In order to explain the difference, we made a simplified version of the trees within the oval highlighted area from Figure 52 and Figure 53. The tree is illustrated in Figure 59. It must be noted that Figure 59 shows a simplified tree and does not represent the tree as 1-to-1 conversion and does not represent the actual number of paths that pass each node. This was deliberate because the purpose of this Figure is to show the difference between considering each action with/without its subject.

Figure 59 shows only 1 action as the source action from Component level 3, the 'Hire_to_attack' action. The top area has 5 'Hire_to_attack' nodes because those nodes had different attacker has different attacker, but the same target (thief1, the target of Component 3). The bottom area only has 1 'Hire_to_attack' node. At first

glance both top and bottom areas might not have much different set of actions, but that was not the case.

For the top area, each 'Hire_to_attack' node had different set of next possible actions. The 'Hire_to_attack' that hired soldier1 as attackers could only be followed by 'Goto' or 'Hire_to_attack' node. The 'Hire_to_attack' node that hired Solider3 could be followed by 'Goto' or 'Attack.' It can be seen that hiring different NPCs would result in different situations where the player needed different actions to complete the quest.

If we considered only player's actions, but not the subject of the actions, it meant we must select only 1 'Hire_to_attack' node as the first node, amongst the nodes in the top area. If we selected the one with soldier3 as attacker, the other 'Hire_to_attack' would be discarded because they would be regarded as duplicates when they were discovered during the query. This would result in lower number of unique paths. Instead of 4 possible actions ('Goto', 'Direct_Attack', 'Hire_to_Attack' and 'No_Action_Needed') to complete the quest, we would only have 2 actions (Goto & Attack). This was why it was important to consider the subject of actions when determining which paths were duplicates.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

6. Summary

In this thesis, we replaced the action-based part of structural rules quest generation with state-based Component to make our quest compatible with dynamic environment games. However, the quest generation procedure was kept the same as in grammar structure quest generation procedure. This new structure was supported by The Token system and Full Condition State system which assigned related and coherent objective to each node / section of a quest. The Quest Query system was able to act as a dynamic environment simulation for the generated quest. The Quest Query system was also capable of verifying if the generated quest could be completed, and capable of discovering a complex and/or interesting path. Generated paths could also be analysed by users.

The state-base system not only making the quest compatible with dynamic environment, it also improved the flexibility of the generated quest and provided developers with greater control and more detailed information of the generation. Players were no longer forced to perform specific action to trigger the quest's completion flag in order to proceed with the quest. By maintaining the quest's frame and structure generation procedure from the Grammar structure, the Grammar structure's advantages were preserved, allowing the quest to be generated with great variety in quest types and maintaining coherence between each step of the quest.

The Token system allowed an object to be assigned into a quest with continuity and consistent manner. Objects did not have to be directly next to each other and could be assigned according to the type of Components they were assigned to. Quest contents were converted into objectives using Full Condition State system to make them compatible with the new state-based structure.

We found interesting sequences of actions which completed a quest in some unexpected manners, such as hiring an NPC to commit a crime inside the palace so that a soldier, who the player must talk with to complete the quest, captured the NPC and escorted the NPC to the prison. This allowed the player to talk to the soldier without having to access the palace. This proved that the system was capable of using the dynamic nature of the simulation to complete the quest.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

The MPQ-Generator framework could be adapted for any dynamic environment game as long as the game did not rely on player-environment interaction (such as dynamic physics that allow the player to kill another NPC by causing accidents). MPQ-Generator's simulator could be used to generate quest or used as benchmark to test how the dynamic environment of that game could be used to complete a designed quest. For benchmarking, the user only had to manually create sets of objectives (Full Condition States) for that quest and query that quest in the simulation (that had been modified to match the tested game). However, some adjustment to input/output to the Quest Query system must also be done.

The quest structure generation and Full Condition State could be modified to generate quests compatible with the game MPQ-Generator was implemented for. The grammar should be modified to only generate valid quest content. This could be achieved by reducing the weight of invalid <Component>s or Components that were not supported to zero. Then the Token system should be modified accordingly so that the output objectives from mini-token and Component pair matched the game structure and intended narrative. The Restriction State system must also be updated to match the new Component's effects. Likewise, the simulator's actions and reactions could be added/removed/modified to match the rule of the game. Additional adjustments or mechanics such as NPC's life cycle could be applied to the simulator to simulate the game as close as possible.

The actions and reactions used in the simulation were only from a small set of possible actions and reactions. The generation still had a performance issue. Changing to a configuration with 4 sequence of actions, the number of paths only increased by around 10%, but the time required to query a Component increased from 1 minute to 15 minutes on average. This accumulated into the total time from around 1 hour to 12 hours to generate a quest. We speculated that more complex and interesting path could be found if higher number of actions is used. Further study is needed to improve the performance so that more objects, actions and reactions can be added too.

The generated paths still contained unnecessary actions that were not needed to complete the quest and some identical set of actions with slightly different parameters and sequences still appeared in some cases. Better filtering criteria are needed to lower these elements.

7. Future Work

Further study should focus on the improvement of the performance of the query system. As more complex worlds with higher number of objects and actions are introduced, the number of possible paths that must be queried increases exponentially. If the system is to be implemented into a commercial game and used in real-time situation, at least the generation time should be less than the time it took for a player to complete the quest and receive the next quest.

One possible approach in improving the performance is to apply Dynamic Programming on GameState. When a new GameState is queried, all results of that query (from all combinations of sequence of actions) will be recorded into a database. Then during the next query, if the input GameState matches the recorded GameState, no actual exhaustive search will be carried out. The system will then look at all recorded GameStates, select those that satisfy the input GoalState, and return the recorded [path].

This means that any gameState that has been queried will no longer need to be calculated ever again. Thus, the longer the system runs, the better the performance will be. In order to implement this a proper database system is needed to maintain all the recorded GameStates and all of their results and combinations of actions.

Another possible further study is to try to implement a more generalised system and structure for the MPQ-Generator. The current structure is a simplified abstract representation of real player actions. If the system is to be implemented into a game, a flexible input/output of configuration and other necessary variables between the MPQ-Generator and the game system are required. This includes Streamlining the action, object, reaction, <Component>, etc. to be modular which can be turned-on/off according to the game's environment.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

REFERENCES

[1] *The Elder Scrolls: Skyrim site*. Available: https://elderscrolls.bethesda.net/en/skyrim?, [Accessed: 15 May 2018]

[2] *Fallout 4 site*. Available: https://fallout4.com/, [Accessed: 15 May 2018]

[3] S. Tosca, "The Quest Problem in Computer Games " in *Proceedings of the Proceedings of Technologies for Interactive Digital Storytelling and Entertainment conference (TIDSE),* Darmstadt, 2003.

[4] F. K. C. Santos and G. L. Ramalho, "A Parametric Analysis and Classification of Quests in MMORPGs," in *Proceedings of the SBC - Proceedings of SBGames 2012,* Brasilia – DF – Brazil, 2012.

[5] *Spelunky site*. Available: http://www.spelunkyworld.com/, [Accessed: 15 May 2018]

[6] *Spelunky jungle area map 1* [Online]. Available: http://spelunky.wikia.com/wiki/The_Jungle/HD, [Accessed: 10 March 2018]

[7] *Spelunky jungle area map 2* [Online]. Available: http://spelunky.wikia.com/wiki/Black_Market, [Accessed: 10 March 2018]

[8] *Spelunky jungle area map 3* [Online]. Available: https://www.reddit.com/r/spelunky/comments/521rql/looking_for_highres_images_of_full_levels/), [Accessed: 10 March 2018]

[9] *Spelunky jungle area map 4* [Online]. Available: http://spelunky.wikia.com/wiki/Restless_Dead_level, [Accessed: 10 March 2018]

[10] B. Kybartas and R. Bidarra, "A Survey on Story Generation Techniques for Authoring Computational Narratives," in *Proceedings of the IEEE Transactions on Computational Intelligence and AI in Games,* 2017.

[11] *Neo-Scavenger Site.* Available: http://bluebottlegames.com/games/neo-scavenger [Accessed: 15 May 2018]

[12] *Renowned Explorers site*. Available: http://renownedexplorers.com/, [Accessed: 15 May 2018]

[13] *Dwarf Fortress site*. Available: http://www.bay12games.com/dwarves/, [Accessed: 15 May 2018]

[14] *Rimworld site*. Available: https://rimworldgame.com/, [Accessed: 15 May 2018]

[15] *The Witcher 3 Site*. Available: http://thewitcher.com/en/witcher3, [Accessed: 15 May 2018]

[16] *Nier:Automata site*. Available: https://www.niergame.com/gb/, [Accessed: 15 May 2018]

[17] *World of Warcraft site*. Available: https://worldofwarcraft.com/en-us/, [Accessed: 15 May 2018]

[18] *Mount & Blade site*. Available: https://www.taleworlds.com/, [Accessed: 15 May 2018]

[19] *MIddle Earth: Shadow of Mordor site*. Available: https://www.warnerbros.com/videogame/middle-earth-shadow-mordor [Accessed: 15 May 2018]

[20] *Neo Scavenger world map* [Online]. Available: http://bluebottlegames.com/games/neo-scavenger, [Accessed: 15 May 2018]

[21] *Spelunky mine area* [Online]. Available: http://www.spelunkyworld.com/images/spelunky-ss01.jpg [Accessed: 15 May 2018]

[22] *A example campaign map of Renowned Explorers: International Society* [Online]. Available: http://renownedexplorers.com/#screenshot-ec-2 [Accessed: 15 May 2018]

[23] *A gameplay screenshot of The Witcher 3: Wild Hunt* [Online]. Available: https://gamesdb.launchbox-app.com/games/images/15977, [Accessed: 15 May 2018]

[24] *A gameplay screenshot of Nier:Automata* [Online]. Available: https://www.niergame.com/gb/ [Accessed: 15 May 2018]

[25] *A gameplay screenshot of World of Warcraft* [Online]. Available: http://www.pcgamer.com/what-we-want-from-world-of-warcraft-in-2017/ [Accessed: 15 May 2018]

[26] *A gameplay screenshot of Dwarf Fortress* [Online]. Available: http://

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

www.bay12games.com/dwarves/screens/adv1.html, [Accessed: 15 May 2018]

[27] *A gameplay screenshot of moded Dwarf Fortress* [Online]. Available: https://www. polygon.com2014775877073dwarf-fortress-3d-mod [Accessed: 15 May 2018]

[28] *A gameplay screenshot of RimWorld* [Online]. Available: https:// rimworldgame.com/images/screens/megacolony.jpg, [Accessed: 15 May 2018]

[29] *A gameplay screenshot of Mount & Blade: Warband* [Online]. Available: https:// www.heypoorplayer.com/2016/10/01/mount-blade-warband-gets-feature-video/ [Accessed: 15 May 2018]

[30] *A gameplay screenshot of Middle-earth: Shadow of Mordor* [Online]. Available: https://segmentnext.com/2014/09/30/understanding-nemesis-system-in-middle-earth-shadow-of-mordor/, [Accessed: 15 May 2018]

[31] *A gameplay screenshot of The Elder Scrolls: Skyrim* [Online]. Available: https:// www.digitaltrends.com/game-reviews/the-elder-scrolls-v-skyrim-review/, [Accessed: 15 May 2018]

[32] *A gameplay screenshot of Fallout 4* [Online]. Available: https:// www.gamecrate.com/fallout-4-radpacks-horrors-commonwealth/13791 [Accessed: 15 May 2018]

[33] J. Doran and I. Parberry, "A prototype quest generator based on a structural analysis of quests from four MMORPGs," in *Proceedings of the Second International Workshop on Procedural Content Generation in Games,* Bordeaux France, 2011.

[34] J. Grey and J. J. Bryson, "Procedural quests: A focus for agent interaction in role-playing-games," presented at the Proceedings of the AISB 2011 Symposium: AI & Games, 2011. Available: http://opus.bath.ac.uk/27232/

[35] GVMERS. (2017). *The Rise and Fall of S.T.A.L.K.E.R.* | *Documentary* [Online]. Available: https://www.youtube.com/watch?v=rYNjcM7wCy8), [Accessed: 12 March 2018]

[36] E. S. d. Lima, B. Feij, and A. L. Furtado, "Hierarchical generation of dynamic and

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

nondeterministic quests in games," in Proceedings of the 11th Conference on Advances in Computer Entertainment Technology, Funchal, Portugal, 2014.

[37] Y.-S. Lee and S.-B. Cho, "Dynamic quest plot generation using Petri net planning," in *Proceedings of the Proceedings of the Workshop at SIGGRAPH Asia,* Singapore, Singapore, 2012.

[38] D. B. Buss, M. V. Eland, R. Lystlund, and P. Burelli, "The Quality System - An Attempt to Increase Cohesiveness Between Quest Givers and Quest Types," in *8th International Conference on Interactive Digital Storytelling* Copenhagen, Denmark, 2015, pp. 381-384: Springer International Publishing, cham.

[39] A. Machado, P. Santos, and J. Dias, "On the Structure of Role Playing Game Quests," *Revista de Ciências da Computação,* p. 18, 2017.

[40] C. Ashmore and M. Nitsche, "The Quest in a Generated World," presented at the Proceedings of DiGRA 2007 Conference, 2007.

[41] J. Valls-Vargas, S. Ontañón, and J. Zhu, "Towards story-based content generation: From plot-points to maps," in *Proceedings of the 2013 IEEE Conference on Computational Inteligence in Games (CIG)*, Niagara Falls, ON, 2013. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6633654&isnumber=6633607

[42] I. Khaliq and Z. Watson, "The Omni Framework: A Destiny-Driven Solution to Dynamic Quest Generation in Games," in *2018 IEEE Games, Entertainment, Media Conference (GEM)*, Galway, 2018, pp. 306-311.

[43] J. v. d. Water, "A Framework for Formalizing Dynamic Quests," Master Thesis, Department of Computer Science, Utrecht University, 2011.

[44] A. Stocker and C. Alvin, "Non-Linear Quest Generation," in *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference (FLAIRS-31),* 2018. Available: https://www.aaai.org/ocs/index.php/ FLAIRS/ FLAIRS18/paper/viewPaper/17606

[45] A. M. Sullivan, "The Grail Framework: Making stories playable on three levels in CRPGS," Ph.D., Computer Science, University of California, Santa Cruz, 2012.

[46] A. Sullivan, A. Grow, M. Mateas, and N. Wardrip-Fruin, "The design of Mismanor: creating a playable quest-based story game," in *Proceedings of the International Conference on the Foundations of Digital Games*, Raleigh, North Carolina, 2012, pp. 180-187: ACM.

[47] B. Kybartas and C. Verbrugge, "Analysis of ReGEN as a Graph-Rewriting System for Quest Generation," in *Proceedings of the IEEE Transactions on Computational Intelligence and AI in Games,* 2014.

[48] V. Breault, S. Ouellet, and J. Davies, "Let CONAN tell you a story: Procedural quest generation," 2018. arXiv:1808.06217

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University

VITA

NAME	Thongtham Chongmesuk
DATE OF BIRTH	17 July 1993
PLACE OF BIRTH	Bangkok
INSTITUTIONS ATTENDED	Bachelor of Engineering in Computer Engineering, Chulalongkorn University 2016
HOME ADDRESS	45 ซอยเพชรเกษม 50 ถนนเพชรเกษม แขวงบางหว้า เขตภาษีเจริญ กทม. 10160
PUBLICATION	T. Chongmesuk, V. Kotrajaras, “Multi-Paths Generation for Structural Rule Quests,” in Proceeding of the 16th International Joint Conference on Computer Science and Software Engineering (JCSSE2019), Pattaya, Thailand, 2019, pp. 97-102.

จุฬาลงกรณ์มหาวิทยาลัย
Chulalongkorn University